



SDN-μSense

Project No. 833955

Project acronym: SDN-microSENSE

Project title:

SDN - microgrid reSilient Electrical eNergy SystEm

Deliverable D4.5

Blockchain-based Energy Trading Framework

Programme: H2020-SU-DS-2018

Start date of project: 01.05.2019

Duration: 36 months

Editor: CERTH

Due date of deliverable: 31/12/2020

Actual submission date: 30/12/2020



Deliverable Description:

| | |
|--------------------------|---|
| Deliverable Name | Blockchain-based Energy Trading Framework |
| Deliverable Number | D4.5 |
| Work Package | WP 4 |
| Associated Task | T4-5 |
| Covered Period | M05-M20 |
| Due Date | M20 |
| Completion Date | M20 |
| Submission Date | 23/12/2020 |
| Deliverable Lead Partner | CERTH |
| Deliverable Author(s) | CERTH, TECNALIA, IREC, IDENER, IPTO |
| Version | 1.0 |

| Dissemination Level | | |
|---------------------|---|----------|
| PU | Public | X |
| PP | Restricted to other programme participants (including the Commission Services) | |
| RE | Restricted to a group specified by the consortium (including the Commission Services) | |
| CO | Confidential, only for members of the consortium (including the Commission Services) | |

CHANGE CONTROL
DOCUMENT HISTORY

| Version | Date | Change History | Author(s) | Organisation |
|---------|------------|---|--|-----------------|
| 0.1 | 13/11/2020 | Table of Contents | Vakakis Nikolaos, Pasias Achilleas, Kotsiopoulos Athanasios (CERTH) | CERTH |
| 0.2 | 20/11/2020 | State of the art in blockchain-based energy trading systems completed | Jose Antonio Perez Jimenez (IDENER), Panagiotis Famelis (IPTO) | IDENER, IPTO |
| 0.3 | 30/11/2020 | Requirements analysis section completed | Vakakis Nikolaos, Lazaridis Georgios (CERTH), Marisa Escalante Martinez, Inaki Seco Aguirre, Dediego Santiago (TECNALIA) | CERTH, TECNALIA |
| 0.4 | 05/12/2020 | All sections related to Blockchain-based Intrusion and Anomaly Detection module completed | Marisa Escalante Martinez, Inaki Seco Aguirre, | TECNALIA |

| | | | | |
|-----|------------|--|--|-----------------------------|
| | | | Dediego Santiago (TECNALIA) | |
| 0.5 | 09/12/2020 | Development section of e-auction module completed | Vakakis Nikolaos, Kotsiopoulos Athanasios (CERTH) | CERTH |
| 0.6 | 14/12/2020 | First draft for peer review | Vakakis Nikolaos, Kotsiopoulos Athanasios, Lazaridis Georgios, Pasias Achilleas (CERTH), Marisa Escalante Martinez, Inaki Seco Aguirre, Dediego Santiago (TECNALIA), Jose Antonio Perez Jimenez (IDENER), Panagiotis Famelis (IPTO), Pol Paradel Sola (IREC) | CERTH, TECNALIA, IPTO, IREC |
| 0.7 | 23/12/2020 | Second peer review version, addressing the comments from initial peer review | Vakakis Nikolaos (CERTH), Marisa Escalante Martinez (TECNALIA), Panagiotis Famelis (IPTO), Pol Paradel Sola (IREC) | CERTH, TECNALIA, IPTO, IREC |
| 1.0 | 29/12/2020 | Final version, after receiving and addressing the reviewer's comments | Vakakis Nikolaos (CERTH) | CERTH |

DISTRIBUTION LIST

| Date | Issue | Group |
|------------|--------------------------|------------------------------------|
| 16/12/2020 | 1 st Revision | Energynautics, SINTEF, TM, QM, SAB |
| 23/12/2020 | 2 nd Revision | Energynautics, SINTEF, TM, QM, SAB |
| 24/12/2020 | Acceptance | Energynautics |
| 29/12/2020 | Acceptance | SINTEF, TM, QM |
| 30/12/2020 | Acceptance | SAB |
| 30/12/2020 | Submission | AYESA |

SAB APPROVAL

| NAME | INSTITUTION | DATE |
|-------------------------|-------------|------------|
| Prof. Sokratis Katsikas | NTNU | 30/12/2020 |
| Marc Stauch | LUH | 30/12/2020 |
| Dave Raggett | ERCIM | 30/12/2020 |

Academic and Industrial partner revision

| NAME | INSTITUTION | DATE |
|---------------------|--------------------------------------|------------|
| Ellen Krohn Aasgård | Academic partner: SINTEF | 29/12/2020 |
| Nis Martensen | Industrial partner: Energynautics | 21/12/2020 |

Quality and Technical manager revision

| NAME | INSTITUTION | DATE |
|-----------------------|-------------|------------|
| Dimosthenis Ioannidis | CERTH | 27/12/2020 |
| Drosou Anastasios | CERTH | 27/12/2020 |

Table of contents

| | |
|---|-----------|
| Table of contents | 5 |
| Acronyms..... | 7 |
| List of Figures..... | 8 |
| List of Tables..... | 9 |
| Executive Summary..... | 11 |
| 1. Introduction | 12 |
| 1.1 Purpose of this document..... | 12 |
| 1.2 Relation to other tasks and deliverables | 12 |
| 1.3 Structure of this document | 12 |
| 2. State of the art in blockchain-based energy trading systems | 13 |
| 3. Architecture and Requirements | 16 |
| 3.1 Objective of the system and placement in SDN-microSENSE architecture | 16 |
| 3.2 Requirements analysis | 17 |
| 3.2.1 Major inputs and outputs | 17 |
| 3.2.2 Functional requirements..... | 18 |
| 3.2.3 Non-functional requirements | 20 |
| 4. System analysis..... | 23 |
| 4.1 Component model | 25 |
| 4.1.1 E-auction module | 25 |
| 4.1.1.1 E-auction mechanism..... | 26 |
| 4.1.1.2 Penalty mechanism..... | 30 |
| 4.1.1.3 Vickrey-Clarke-Groves chaincode | 33 |
| 4.1.1.4 ERC20 token chaincode..... | 38 |
| 4.1.1.5 Priority table chaincode | 40 |
| 4.1.1.6 Market chaincode | 42 |
| 4.1.1.7 E-auction module APIs | 43 |
| 4.1.2 Blockchain Based Intrusion and Anomaly Detection module..... | 57 |
| 4.2 Interfaces model | 62 |
| 4.2.1 External interfaces with other SDN-microSENSE application plane modules | 62 |
| 4.2.1.1 Communication of e-auction module with EMO | 62 |
| 4.2.1.2 Communication of e-auction module with S-RAF..... | 64 |

| | | |
|-----------|--|------------|
| 4.2.1.3 | Communication of Blockchain-based Intrusion and Anomaly Detection with XL-SIEM | 64 |
| 4.2.2 | External interfaces with SDN-microSENSE infrastructure plane modules..... | 65 |
| 4.2.2.1 | Connection of e-auction module with RPI of smart meter..... | 65 |
| 4.2.2.2 | Connection of Blockchain Based Intrusion and Anomaly Detection module with RPI of smart meter | 67 |
| 5. | System verification..... | 68 |
| 5.1 | E-auction module unit tests | 69 |
| 5.2 | Blockchain Based Intrusion and Anomaly Detection module unit tests..... | 116 |
| 6. | System Installation..... | 131 |
| 6.1 | E-auction module | 131 |
| 6.1.1 | E-auction module installation | 131 |
| 6.2 | Blockchain Based Intrusion and Anomaly Detection module..... | 132 |
| 6.2.1 | BIAD installation..... | 132 |
| 6.2.2 | Agent installation | 133 |
| 7. | Innovation Summary | 133 |
| 8. | Conclusions | 135 |
| | References | 137 |

Acronyms

| Acronym | Explanation |
|---------|--|
| AA | Adaptive Aggressiveness |
| API | Application Programming Interface |
| BIAD | Blockchain-based Intrusion and Anomaly Detection |
| CDA | Continuous Double Auction |
| DER | Distributed Energy Resource |
| DLT | Distributed Ledger Technologies |
| DSO | Distribution System Operator |
| DHT | Distributed Hash Table |
| EC | Energy Contribution |
| EPES | Electrical Power Energy System |
| ESCO | Energy Service Company |
| ESS | Energy Storage System |
| EV | Electric Vehicle |
| GUID | Globally Unique Identifier |
| ISO | Independent System Operator |
| IDPS | Intrusion Detection and Prevention System |
| LCOE | Levelized Cost of Energy |
| LV | Low Voltage |
| MSP | Membership Service Provider |
| P2P | Peer to Peer |
| PV | Photovoltaic |
| QoS | Quality of Service |
| SDK | Software Development Kit |
| SPoF | Single Point of Failure |
| TLS | Transport Layer Security |
| TRL | Technology Readiness Level |
| TSO | Transmission System Operator |
| UUID | Universally Unique Identifier |
| ZIP | Zero Intelligence Plus |

List of Figures

| | |
|--|----|
| Figure 1: Placement of blockchain-based energy trading system within SDN-microSENSE architecture | 17 |
| Figure 2: e-auction Fabric network architecture | 26 |
| Figure 3: VCG auction mechanism demonstration with 3 winners | 27 |
| Figure 4: VCG auction mechanism demonstration with 1 winner | 27 |
| Figure 5: VCG auction mechanism demonstration, winners defined by priority of bidding | 28 |
| Figure 6: Auctioneer-Bidder API endpoints | 44 |
| Figure 7: /apply request parameters and responses | 45 |
| Figure 8: /getBalance request parameters and responses | 46 |
| Figure 9: /transfer request parameters and responses | 47 |
| Figure 10: /initVickreyCGAuction request parameters and responses | 48 |
| Figure 11: /getVickreyCGAuction request parameters and responses | 48 |
| Figure 12: /VCGsubmitSealedBid request parameters and responses | 49 |
| Figure 13: /VCGreveal request parameters and responses | 50 |
| Figure 14: /VCGaward parameters and responses | 50 |
| Figure 15: /VCGsettle request parameters and responses | 51 |
| Figure 16: /getMarkets request responses | 51 |
| Figure 17: /applyToMarket request parameters and responses | 52 |
| Figure 18: /getPriorityTable request responses | 52 |
| Figure 19: ESCO API endpoints | 53 |
| Figure 20: /initToken request parameters and responses | 54 |
| Figure 21: /getApplicants request parameters and responses | 55 |
| Figure 22: /updateBalance request parameters and responses | 56 |
| Figure 23: /initMarket request responses | 57 |
| Figure 24: /initPriorityTable request responses | 57 |
| Figure 25: Blockchain Based Intrusion and Anomaly Detection system components | 59 |
| Figure 26: BIAD behaviour model | 62 |
| Figure 27 Example request of Market Validator | 63 |
| Figure 24: Agent algorithm | 66 |
| Figure 25: Penalty paying algorithm | 67 |

List of Tables

| | |
|---|-----|
| Table 1: List of functional requirements covered by the Blockchain-based Energy Trading System.... | 18 |
| Table 2: List of non-functional requirements covered by the Blockchain-based Energy Trading System | 20 |
| Table 3: Permissioned vs permissionless blockchain networks..... | 23 |
| Table 4: Comparative analysis of different blockchain platforms | 23 |
| Table 5: Sample priority table | 30 |
| Table 6: Sample priority table with prosumer 1 being imposed 50 tokens of penalty | 31 |
| Table 7: Sample priority table with prosumer 1 being imposed 50 tokens of penalty and prosumer 4 being imposed 100 tokens of penalty..... | 31 |
| Table 8: Sample priority table with prosumer 1 having paid 50 tokens of penalty and prosumer 4 being imposed 100 tokens of penalty..... | 31 |
| Table 9: Vickrey-Clarke-Groves chaincode functions | 33 |
| Table 10: Vickrey Clarke-Groves auction structure | 34 |
| Table 11: Participant structure | 36 |
| Table 12: Commodity structure | 36 |
| Table 13: Result structure | 36 |
| Table 14: HiddenBid structure | 37 |
| Table 15: RevealedBid structure | 37 |
| Table 16: ERC20 token chaincode functions..... | 38 |
| Table 17: Token structure | 40 |
| Table 18: Priority table chaincode functions | 40 |
| Table 19: Priority Table structure | 41 |
| Table 20: Market chaincode functions | 42 |
| Table 21: Market structure | 43 |
| Table 22 Functions from the chaincode..... | 59 |
| Table 23 Device structure | 60 |
| Table 24 Hash structure | 60 |
| Table 25 Record structure..... | 61 |
| Table 26 Alert structure | 61 |
| Table 27: EMO – e-auction interface | 62 |
| Table 28: e-auction – SRAF interface | 64 |
| Table 29: BIAD - XL-SIEM interface | 65 |
| Table 30: E-AUCTION_01 unit test..... | 69 |
| Table 31: E-AUCTION_02 unit test..... | 82 |
| Table 32: E-AUCTION_04 unit test..... | 93 |
| Table 33: E-AUCTION_05 unit test..... | 111 |
| Table 34: BIAD_01 unit test | 116 |
| Table 35: BIAD_02 unit test | 118 |
| Table 36: BIAD_03 unit test | 122 |
| Table 37: BIAD_04 unit test | 125 |
| Table 38: Lib_mon_001 unit test | 129 |
| Table 39: Lib_hash_001 unit test..... | 129 |
| Table 40: Fabric_conn_001 unit test | 130 |

| | |
|-------------------------------------|-----|
| Table 41: agent_001 unit test | 131 |
|-------------------------------------|-----|

Executive Summary

This document is a deliverable of the SDN-microSENSE project, funded by the European Commission (EC) under its Horizon 2020 Research and Innovation Programme (H2020).

It comprises deliverable *D4.5, Energy Exchange using Blockchain Technologies* of the SDN-microSENSE project. The developed Blockchain-based Energy Trading Framework is a module of the SDN-SELF component, that utilizes Fabric blockchain technology to achieve secure and sustainable energy transactions between the participants of a permissioned blockchain network. The two modules of the Energy Trading Framework are the e-auction and the blockchain-based Intrusion Detection System. The task receives as input the Functional Use Case Requirements, the Data Protection Requirements, the Data Security Requirements and the Reliability Requirements from task T2.2, as well as the SDN-microSENSE platform specification and architecture from T2.3. The output it produces is the fourth layer of the SDN-SELF component. The document includes the identification of the aforementioned requirements and the analysis of how they are being addressed, as well as a State-of-the-Art Analysis of recent literature, related to the Blockchain technologies and the energy markets. Subsequently, based on these assessments, the Blockchain-based Energy Trading Framework is thoroughly described, including the functionalities, the inputs/outputs and the interconnections with other SDN-microSENSE components and modules.

The framework is based on a permissioned blockchain network, which ensures that energy and financial transaction-related data are not exposed outside the consortium of organizations participating in the network, while it allows them to interact with smart contracts (chaincodes) to perform several actions within the network. The privacy of the participants is achieved by utilizing communication channels between organizations which ensure exposure of transactions and related data only to authorized members of those channels and not to any other network members. Furthermore, the nature of the blockchain technology ensures trusted transactions between the participants and non-repudiation of actions. Additionally, the Energy Trading Framework, receives and utilizes input from SDN-microSENSE security, risk assessment and decision support layers, thus ensuring safety of transactions against cyber-security threats and destabilization of the grid.

The Technology Readiness Level (TRL) of the Energy Trading Framework by the end of the project is expected to be equal to 7, since it will be evaluated in real life operational environment, consisting of a group of prosumers belonging to same DSO, in Use Case 6.

1. Introduction

1.1 Purpose of this document

The purpose of this document is to present the research conducted and the results achieved in Task 4.5, *Energy Exchange using Blockchain Technologies*. The task is included in Work Package 4, *Cyber-secured & Resilient SDN-based Energy Ecosystem*. The blockchain-based energy trading system is a module of the SDN-SELF component which utilizes blockchain technology to host an energy exchange framework, the operation of which takes into account the energy sustainability, the privacy of the participants sensitive data, and the security of their infrastructure.

1.2 Relation to other tasks and deliverables

Task 4.5 receives the security and privacy requirements from task 2.2 and the platform's architecture from task 2.3 and produces the fourth layer of the SDN-SELF component. Deliverable 4.5 interacts with the following deliverables, references to which will be found in this document:

- D2.2, User & Stakeholder, Security and Privacy Requirements: This deliverable is the output of Task 2.2 and defines the user (energy operators, prosumers, consumers etc.), the security and the privacy requirements of the SDN-microSENSE project.
- D2.3, Platform Specification & Architecture: This deliverable is the output of Task 2.3, which defines the SDN-microSENSE architecture fed by the energy application requirements in security, privacy, and information protection.
- D3.5, Implementation of SDN-microSENSE Risk Assessment Framework: This deliverable is the output of Task 3.5 and assesses the level of risk in all the involved EPES devices and systems and forwards the data to the Energy Trading Framework.
- D4.4, Distribution Grid Restoration using Deterministic Optimisation and Machine Learning-based Threat Handling: This deliverable is the output of Task 4.4 and deploys the appropriate management processes that will balance the energy supply and demand for each island. These processes will determine whether the transactions conducted in the Energy Trading Framework endanger the island's stability or not.
- D5.1, Large Scale SIEM & Ultrafast Logging: This deliverable performs effective and timely logging of cybersecurity events in the EPES and provides advanced protection for the involved stakeholders of the Energy Trading Framework.

1.3 Structure of this document

The document is structured in 8 sections as follows:

1. Section 1 is the introduction of the document, and it presents the structure of the document, the purpose of the deliverable, and its relation to other tasks and deliverables.
2. Section 2 includes the State-of-the-Art Analysis of recent literature, related to Blockchain-based trading systems.
3. Section 3 identifies the user, security and privacy requirements and describes how they are addressed by the Energy Trading Framework. Also, it describes the placement of the Energy

Trading Framework within the SDN-microSENSE architecture and documents its major inputs and outputs.

4. Section 4 analyses the functionalities and the interfaces of the two modules of the framework, namely the e-auction module and the Blockchain-based Intrusion and Anomaly Detection (BIAD) module.
5. Section 5 demonstrates the unit tests that were conducted in order to ensure that the two modules' operation is sustainable and efficient and presents their results.
6. Section 6 describes the procedures and the specifications required for the installation of the e-auction module and the BIAD Module.
7. Section 7 presents the innovation summary that this deliverable proposes.
8. Section 8 concludes the document.

2. State of the art in blockchain-based energy trading systems

The latest technological developments are leading us to the transition to decentralised power systems. Microgrids, distributed energy resources (DERs) and micro-generation are transforming traditional consumers to what is called prosumers, i.e., consumers who also produce energy, thus enabling them to participate in the energy grid in new disruptive ways. This new approach introduces new challenges and opportunities in the energy markets. Until now, the energy market access was centralised, usually orchestrated by the System Operator (an ISO or a TSO) and concerned large generation units and clusters of units controlled by bigger parties and corporations making it difficult for small scale prosumers to actually participate in the energy market. By developing decentralized markets and enabling prosumers to trade among themselves and on the edge of the grid, a more robust and cleaner system can exist. In the technical and academic literature, new decentralised approaches have been developed to tackle this challenge.

One of the most recurring technique is the use of blockchain-based energy trading systems. On this aspect, distributed ledger technologies (DLT) such as blockchain technologies enable the decentralised, trusted, and secure exchange of energy trading transactions between parties. The blockchain systems' decentralised nature enhances system resilience to outages and cyberattacks, avoiding the single point of failure design of many current centralised systems. Additionally, the data in blockchain systems is easily verifiable by interested parties and always available, enhancing data integrity and accounting. All of these features, along with the automation provided by the introduction of smart contracts in the blockchain machinery, establishes this technology as a basis for the future energy sector [1]. On the context of Task 4.5 of this project, several technical implementations have been carried on new blockchain-based energy trading technologies and tools. On this aspect, the current state of the art in this field is presented in the upcoming paragraphs.

In 2017, Wang et al. [2] proposed a peer-to-peer decentralised electricity transaction mode for microgrids based on Bitcoin [3] as a blockchain and currency system and the use of the continuous double auction (CDA) mechanism. Additionally, this work applies an adaptative aggressiveness (AA) trading strategy for increasing profits to market participants. The CDA mechanism is a fixed-duration auction system in which several buyers and sellers bid on the market to buy and sell goods (in this case,

energy), respectively, and where transactions will take place at any time if an offer to buy and an offer to sell match [4]. Based on this transaction pattern, prosumers will send quotes to the CDA market according to an AA strategy throughout this transaction pattern and adjust quotes continuously according to transaction results. Buyers and sellers achieve digital proof of energy transaction and settlement of contracts using blockchain once the market matches, thereby achieving direct microgrid electricity transactions. Bitcoin is used for currency exchange, and digital certificates are employed as contracts that indicate the right to use the corresponding electricity once the auction ends.

Later, A. Hahn [5] et al. demonstrated the use of blockchain-based energy auctions through the Ethereum blockchain technology [6]. The auction mechanisms implemented through smart contract is the Vickrey second price auction [7] mechanism to encourage honest bids on auctions. This work had a real successful implementation on a testbed with a 72kW photovoltaic (PV) array. The architecture of this work is based on multiple agents in charge of selling and bidding operations, including smart meter agents and the respective agent for smart contract execution.

In 2018, Guerrero et al. [8] introduced the first technique for ensuring network constraints in P2P blockchain-based trading schemes in low-voltage (LV) networks. This work is based on a CDA auction mechanism with a zero intelligence plus (ZIP) strategy implemented as agents. ZIP agents use an adaptive process that can offer output somewhat close to that of stock exchange traders. The last key aspect of this work is the Network Permission Structure. This mechanism validates the auctions' voltage variations and line congestions to prevent problems in the network while informing the market participants of potential issues. Moreover, the transaction system internalises the extra cost due to the technical constraints of the network. The benchmarks carried on this work indicate that this scheme does not suffer from scalability issues which is a recurring issue of blockchain-based environments.

Furthermore, a new blockchain-based energy trading scheme is presented in [9]. In this work, Li et al. introduce a new variant with permissioned blockchain technologies to protect privacy and security. This way, having a more restricted blockchain instead of a public one, the attack surface is reduced while preserving privacy to specifically authorised participants in the market. Another interesting addition to preventing the delay in blockchain-based transactions is the introduction of a credit-based payment scheme to support faster and more frequent trading among peers. The removal of the use of cryptocurrency results into an enhancing of performance of the overall energy trading scheme.

Privacy is also the subject of the work in [10]. As Gai et al. discuss one of the main disadvantages of a blockchain solution is its vulnerability in terms of linking attacks. As the transactions are traceable, an attacker that has access to other public data (from utility companies or from public records for example) could link transactions to users even if they are anonymised. They describe three types of attacks and present a way of defend against them. The basic idea revolves around the mapping of different accounts. Basically, every time transactions exceed a certain threshold, the payments are deposited to newly created accounts, which are mapped to the original, making it difficult for the attacker to discover the mappings. The proposal takes into account also the fact that some accounts may be inactive for some time; to preserve their privacy, the system creates new dummy accounts every time a new account is created, proportional to the inactive accounts, every time a transaction is made.

An interesting combination of machine learning and blockchain can be found in [11]. Ferrag et al. propose the Deepcoin framework that uses a byzantine fault tolerance based blockchain together with a deep-learning IDS to detect network attacks and deceitful transactions. They describe a blockchain architecture encompassing home, building and neighborhood networks where prosumers can trade energy amongst themselves. On top of that, they develop a deep learning algorithm that checks if the offers, from sellers or buyers, are genuine, trusted and comply with the set of rules of the system. They also evaluate the Deepcoin framework with different sets, providing accuracy over 90% against most types of attacks.

Wang et al. in [12], propose a trading system that is able to integrate with today's operational models based on crowdsourcing. They categorize between two different classes of energy trading transaction, between prosumers or crowdsources and the utility, as they exist today and among crowdsources. The algorithm minimizes costs by rescheduling any shapeable loads and DERS. In the first phase it behaves akin to day-ahead scheduling, which is the form most markets work today. However, in the second phase any balances that need to be made are first discovered in the local trading market. They also propose different pricing schemes for each type of transactions, on the one hand providing incentives to prosumers to participate in the balancing market and allowing to negotiate prices among themselves. Their proposal in most cases require a central authority to manage the grid and clear the market, however island use cases are described where small-scale energy trading exists without any central authority.

Of course, the blockchain P2P markets are not without problems. Hoarding of energy can be observed, with prosumers buying energy when it is cheaper and keeping it to influence the market prices. This is the problem [13] tries to tackle by providing a demurrage mechanism. Yashaya et al. describe a market model where they try to optimize consumption by shifting demand to off-peak hours and incentivize nearby prosumers to buy on DERS' peak hours. This is done by changing the price of energy with time, so when a prosumers hoards energy or delays to buy time, the buying price will be higher and the selling price lower than the utility's price. Therefore, for a prosumer to increase selling price, they should increase their consumption and to decrease the buying price, they should decrease their consumption. To implement this they use smart contracts, not just as algorithm for the trading itself but trying to incorporate in them any contractual clauses, regulatory guidelines etc. According to their results the demurrage mechanism proves useful as the system produces better prices overall and optimizes consumption when prosumers are active in the local energy market.

The use of smart contracts in blockchain energy trading frameworks is examined also in [14]. Han et al. propose a multidimensional blockchain framework, fully describing the whole process by presenting three layers (infrastructure, players, processes) and what each encompasses. The trading itself follows a double auction where the producers' bids are sorted in increasing order and the consumers' in decreasing order. After matching the bids, the price is beneficial for all and closer to what the participants would actually require. Any unmatched bids are sold or bought by the DSO, who is also responsible to balance and correct any imbalances. Their simulations have shown that over 80% of the energy is traded locally in prices that are satisfactory and that the imbalances concern only 2,91% of the total actual energy.

Lastly, there have been some attempts to combine SDN technologies with blockchain markets as can be seen in [15]. Chaudhary et al. concern themselves with an EV (electric vehicles) energy trading and use SDN to improve QoS. Their approach touches the subject of edge networking, by describing a

layered architecture that separates edge and core networks, using SDN to efficiently orchestrate the traffic. Each area, called cloud, comprises a local SDN controller, a transaction server that maintains the transaction records and meets the energy demands of the users, and local aggregators that behave like miner nodes. Additionally, there exists a global controller, which creates the replication mechanism, maintains the global consistency of the database and reduces communication overhead. Their performance analysis shows that the proposal is lightweight and any additional overhead from SDN and computations is kept to a minimum.

Also in [16], an SDN-based energy internet is proposed with attention to privacy. Lu et al. propose an architecture that is based on DHT, to obfuscate the users' privacy through hash indexing, utilizing a bloom filter to screen the electrical energy information. The sellers produce energy and store it in the main energy storage systems and buyers send requests for energy with specific constraints including benefit, cost etc. The system is responsible to match requests to sellers and inform buyers about their options. Because of the use of DHT, the system behaves essentially as a black box preserving the privacy throughout the transaction. SDN in this case, is used more as a design framework for structuring energy internet in a way that separates data/energy, and forward/transaction planes and less as a technology itself, as the proposal is on a prototype abstract level.

3. Architecture and Requirements

3.1 Objective of the system and placement in SDN-microSENSE architecture

The Blockchain-based Energy Trading System is the last layer of SDN-SELF and aims to provide a safe and trustworthy environment for secure energy trading within islanded parts of the smart-grid. A permissioned blockchain network is established between smart-grid organizations of the island, where they are able to take part in e-auctions as buyers or sellers or, in case of energy service company organizations (ESCO), manage the financial transactions through network dedicated tokens. The tokens are based on ERC20 protocol, which defines a set of rules regarding the manner in which the tokens can be transferred, how transactions are approved and how users can access data about a token. The Blockchain-based Energy Trading System, not only offers important features such as verifiability and immutability of transactions, but also leverages important information provided by other SDN-microSENSE application layer components and modules. More specifically, e-auction results are evaluated by the OTSC tool of EMO module, in order to avoid grid destabilization problems that may occur as a result of energy exchange agreements. Furthermore, the Blockchain-based Intrusion and Anomaly Detection module monitors the smart metering equipment of the organizations and produces security-related alerts that are sent to XL-SIEM of XL-EPDS component and subsequently to S-RAF component. The latter assesses the produced alerts and provides that information to the e-auction module, in order to deny participation to the market to organizations that may operate compromised equipment. The placement of the Blockchain-based Energy Trading System within the SDN-microSENSE platform architecture can be observed in Figure 1.

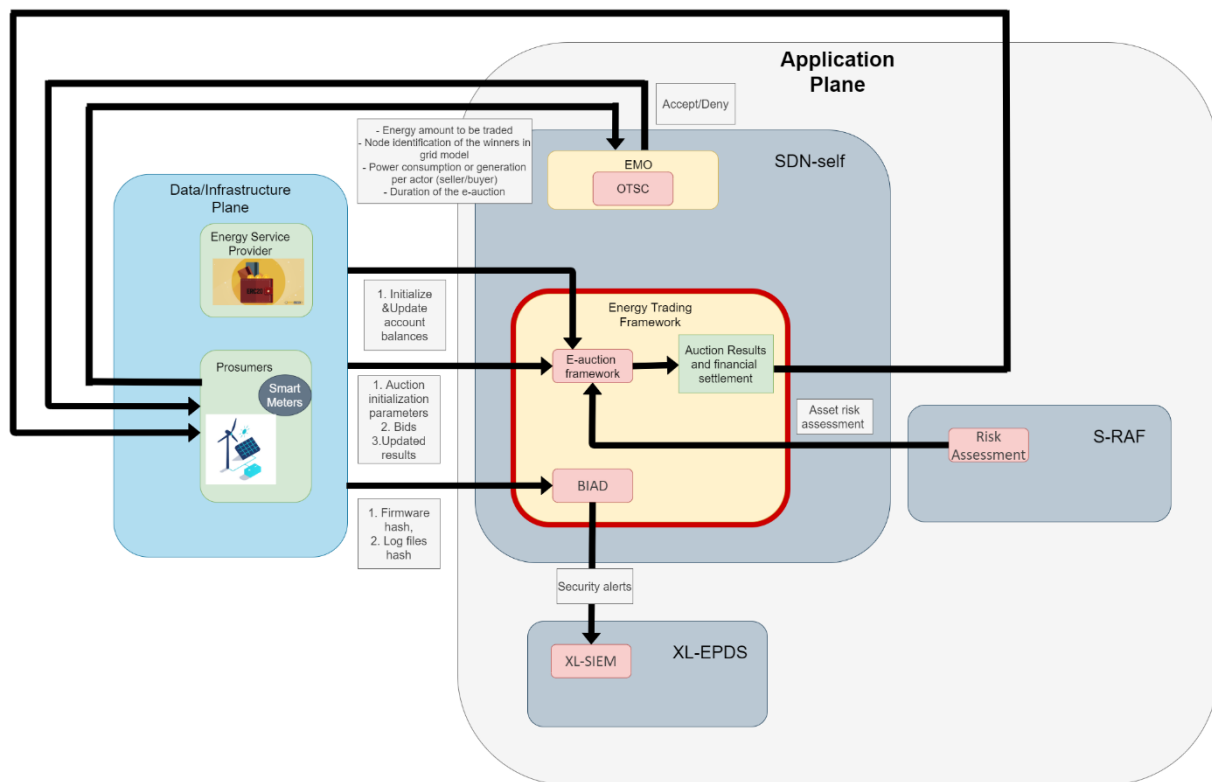


Figure 1: Placement of blockchain-based energy trading system within SDN-microSENSE architecture

3.2 Requirements analysis

3.2.1 Major inputs and outputs

The Blockchain-based Energy Trading System receives input from the data/infrastructure plane and acts according to information collected from different sources of the SDN-microSENSE platform security and decision support layers. More specifically:

- **Receives data related to e-auction processes** from sellers/buyers of energy that participate in the network. The e-auction module implements a specific e-auction protocol that will be discussed later in the deliverable. In order to initialize an e-auction or take part in an existing e-auction, participants have to use the API provided by the e-auction module of the Blockchain-based Energy Trading System.
- **Receives data related to token management** from ESCO, such as token initialization and account initialization data.
- **Receives log, configuration and firmware files** from RPIs connected to smart meters of the prosumers participating in the market, in order to detect possible alterations that could indicate compromise of the devices.
- **Sends smart meter security alerts** that are detected from the Blockchain-based Intrusion and Anomaly Detection module to **XL-SIEM** module of **XL-EPDS**.
- **Receives risk assessment evaluation** of the detected security alerts from the risk assessment module of **S-RAF**.

- **Sends the list of energy transactions** that derive from the e-auction results **to the OTSC tool of EMO module**, which is also part of the **SDN-SELF** component and **receives evaluation of energy trading transactions between prosumers**.
- **Emits e-auction results to be accessible to all the market participants**.

3.2.2 Functional requirements

In Table 1, there is a list with all the functional requirements that are covered by the Blockchain-based Energy Trading System. Those requirements are gathered from D2.2, User & Stakeholder, Security and Privacy Requirements.

Table 1: List of functional requirements covered by the Blockchain-based Energy Trading System

| | |
|--|--|
| FR-UR-06 | The system shall be able to detect and mitigate unauthorised access in near real time (in seconds) and with high accuracy. |
| Involved modules: BIAD, XL-SIEM | |
| The intrusion detection module is able to detect any change to configuration or logs files related with the smart meters that are recorded in the blockchain agent. This system generates an alarm that should be analysed by the XL-SIEM. | |
| FR-UR-17 | The system shall be able to inform in near real time (in seconds) via email and push notification the Security Administrator concerning possible security incidents. |
| Involved modules: BIAD, XL-SIEM | |
| The intrusion detection model, as soon as an access to the configuration or logs files or a failure in the connectivity test is detected, generates an alarm that should be analysed by the XL-SIEM. | |
| FR-GR-06 | The system shall be able to monitor in real time relevant physical parameters at all points of interest in the grid infrastructure. |
| Involved modules: BIAD | |
| The intrusion detection component is available to detect information regarding the usage of RAM and send an alert if this usage is under the threshold established. Also, allows to detect if the smart meter is connected and in case of detecting that it is not connected, generates an alert. | |
| FR-UC4-01 | The system shall perform through encrypted contract (e.g. by using blockchain or similar technology) and protect the privacy of the energy trading parties |
| Involved modules: e-auction | |
| The communication between the participants of the e-auction module is performed through a private Fabric blockchain network. Each participant in the network owns a digital certificate which identifies her/him as a member of the network. In the context of the market, participants are identified by the unique composition of two factors: the Membership Service Provider ID and the Certificate ID (public key) and no other information is exposed. Furthermore, the deployment of the sealed bid Vickrey-Clarke-Groves auction mechanism within the energy trading system, ensures that the bids are cryptographically protected and not revealed to other participants until the bidding deadline. After revealing the bids and performing the Vickrey-Clarke-Groves auction algorithm to award the winners, the results are exposed only to the channel for which members the e-auction is conducted and not to any other network members. | |
| FR-UC4-02 | <ul style="list-style-type: none"> • The system shall provide a specific auctions-oriented protocol to allow an energy trading e-auction (an electronic auction for energy trading between EPES stakeholders) between EPES ecosystem's stakeholders. • The system's energy trading framework shall support cryptographic mechanisms to ensure non-repudiation of bids during an e-auction. |
| Involved modules: e-auction | |

- All the energy transactions conducted are determined by the outcome of the Vickrey-Clarke-Groves auction mechanism, which is a sealed bid type of auction. The auction's algorithms are installed within the chaincode of the blockchain network and they are executed periodically, in determined time slots.
- In a Fabric network the Public Key Infrastructure mechanism is utilized to verify signatures. Each participant signing a transaction cannot state that she/he has not signed the content of this transaction.

| | |
|-----------|---|
| FR-UC4-03 | The system shall support appropriate APIs for the energy trading framework to receive the foreseen energy equilibrium of the islands for a specific timeframe in order to automatically or manually initiate an energy trading auction. |
|-----------|---|

Involved modules: **e-auction**

Each prosumer runs a software agent that communicates with other software tools which provide information regarding the foreseen production and consumption of the building. Based on these information, the prosumers are always aware whether they will participate in an auction as bidders (if they have a deficit of energy), or they will conduct their own auction as auctioneers (if they have a surplus of energy).

| | |
|-----------|---|
| FR-UC4-04 | The system shall ensure the payment of the EPES stakeholder who initiated an energy trading e-auction and the refund of the losing bidders. |
|-----------|---|

Involved modules: **e-auction**

The processes of bid calculation, bid submission and payment of the corresponding stakeholder are performed automatically by the prosumers' agents, therefore the consistency of the participating stakeholders is guaranteed.

| | |
|-----------|---|
| FR-UC4-05 | <ul style="list-style-type: none"> • The system shall monitor the security status of the e-auction participants energy metering equipment. • The system shall deny initialization of an energy e-auction, in case of compromised smart meter of the auctioneer. |
|-----------|---|

Involved modules: **BIAD, XL-SIEM, S-RAF, e-auction**

The Blockchain-based Intrusion and Anomaly Detection module is capable of detecting any intrusion by monitoring the configuration or log files of the gateways connected to smart meters and to check if the smart meters are connected. In case of any intrusion an alert is sent to the XL-SIEM that will provide this information to S-RAF to quantify the risk of the incident and make it accessible for the e-auction module.

| | |
|-----------|---|
| FR-UC4-06 | The system shall adopt a penalty mechanism to confront participation of malicious auctioneers or bidders in the energy trading e-auction. |
|-----------|---|

Involved modules: **e-auction**

In the current version of the Energy Trading System, all the procedures related to the energy market are fully automated. Thus, the stakeholders are not capable to perform any malicious actions that will affect the Framework's consistency or cause damage to other stakeholders. However, in a future extension of the Energy Trading System, users will be granted the capability to perform market-related actions and a penalty mechanism will be necessary. This mechanism is presented in this document.

| | |
|-----------|---|
| FR-UC4-07 | <ul style="list-style-type: none"> • The system shall impose a deadline for bid submission in the energy trading e-auction, after which no more bids will be accepted • The system shall record the energy trading e-auction results in an immutable manner |
|-----------|---|

Involved modules: **e-auction**

The developed chaincode sets strict limitations regarding the permitted time in which the prosumers can submit their bids. No other submissions are accepted if the time limit passes. All the transactions conducted are recorded in the ledger of the blockchain network which provides immutability, due to the nature of blockchain technology.

3.2.3 Non-functional requirements

In Table 2, there is a list with all the non-functional requirements that are covered by the Blockchain-based Energy Trading System. Those requirements are gathered from D2.2, User & Stakeholder, Security and Privacy Requirements.

Table 2: List of non-functional requirements covered by the Blockchain-based Energy Trading System

| ID | Description |
|---|--|
| NFR-SEC-07 | The system administrative interfaces shall be inaccessible to unauthorized users and untrusted parties. |
| Involved modules: BIAD | |
| BIAD uses secured interfaces to communicate with the other tools | |
| NFR-SEC-11 | The system shall be able to use tools to automate configuration and deployment in order to minimize human error. |
| Involved modules: BIAD | |
| BIAD has implemented some mechanisms to minimise as much as possible the human involvement when installing it. But in any case, small involvements of humans are required to configure BIAD prior the installation (see section 6.2) | |
| NFR-SEC-12 | The system shall be able to monitor network devices, operating system, database, firewalls as well as software configurations on a regular basis to ensure that their configuration is not changed by any unauthorized user. |
| Involved modules: BIAD | |
| One of the objectives of BIAD is to detect any change done in the configuration files of any devices through the monitoring of the current state of each device in the blockchain system. | |
| NFR-TST-01 | The system shall be enabled to be automatically testable by software systems (continuous testing). |
| NFR-TST-02 | The system shall be testable by using unit testing, integration testing, security testing, performance testing, system testing, and regression testing procedures. |
| NFR-TST-05 | The system shall be enabled for multi-stage testing, allowing for components to be tested both in isolation and collectively. |
| Involved modules: BIAD, e-auction | |
| BIAD and e-auction once have been deployed are up and running so it is possible to test them, but there is not any test script, so manual testing procedures can be applied. Some of the components, like agent and the chaincode components of BIAD and e-auction, can be tested in an automatic way. The unit tests done are described in the section 5. The other tests (integration, system, etc) will be done in the context of WP7 and WP8. | |

| | |
|---|--|
| NFR-DPT-01 | The system shall be designed to protect personal data by default and maintain this protection throughout the data lifecycle. |
| NFR-DPT-02 | Only the minimum amount of personal data necessary for fulfilling identified purposes shall be processed. |
| NFR-DPT-11 | The system shall be able to store only sensitive data needed and avoid storing unnecessary data. |
| Involved modules: e-auction | |
| <p>All the private data of the participating prosumers are strictly local, and no other prosumers have access to them. Participants are identified by pseudo-identities based on their public key and the Membership Service Provider (MSP) ID. Only the necessary data for the operation of the auction mechanism are being shared during the bidding process (energy request and bid) which are cryptographically protected with commitments before being stored to the e-auction's state. After revealing the bids they are stored to the blockchain state because they are necessary for the calculation of the e-auction results, but the results including the revealed bids are only restrained within the channel for which organizations the market is conducted and not exposed to any other members of the blockchain network.</p> | |
| NFR-DPT-07 | The system shall be inaccessible to unauthorized users so that they cannot access data either intentionally or accidentally. |
| NFR-DPT-08 | The system shall be applying end-to-end encryption for data in transit. It must be ensured that personal data in transit is protected against active (e.g. replays, traffic injection) and passive attacks (e.g. eavesdropping), thus ensuring data integrity. |
| NFR-SEC-09 | The system shall be able to secure the Application Programming Interfaces (APIs). Anonymous access and/or reusable tokens or passwords, clear-text authentication or transmission of content, inflexible access controls or improper authorizations, limited monitoring and logging capabilities must be avoided at all times. |
| NFR-SEC-18 | The system shall be able to use TLS protocol and certifications mechanisms to improve the protection in communication and authentication processes |
| Involved modules: e-auction | |
| <p>The communication between the prosumers is performed within a permissioned Fabric blockchain network, meaning that all the participating prosumers are known and identified with X.509 digital certificates, rather than anonymous. MSPs are used to define the organizations that are trusted by the network members and are also the mechanism that provide members with a set of roles and permissions within the network. Also, the client applications offered by the e-auction module support https protocol for data in transit.</p> | |
| NFR-DPT-09 | The system shall be able to encrypt data when stored, even when data is stored in distributed environments. |
| NFR-DPT-10 | The system shall be able to keep data in unencrypted form only for the duration necessary for the data processing process at hand and no longer neither more extensively. |
| NFR-DPT-22 | The system shall be able to ensure that data are unintelligible except when an authorized individual performs authorized operations on that data. |
| Involved modules: e-auction | |

The submitted bids from the prosumers to the Vickrey-Clarke-Groves chaincode, are encrypted with commitments and only their hash value is stored to the list of submitted bids that is maintained in the e-auction state. Only after, the bidding deadline the bids are revealed and stored in the e-auction state, in order to be available for the Vickrey-Clarke-Groves algorithm, that receives the revealed bids and produces the e-auction results.

| | |
|------------|--|
| NFR-DPT-12 | The system shall be able to ensure that all random numbers, random file names, random GUIDs and random strings are generated in a cryptographically strong fashion. Moreover, the system shall be seeding the random algorithms with sufficient entropy. |
|------------|--|

Involved modules: **e-auction**

When a priority table, a new e-auction or a new market are initiated, a UUID is generated that uniquely identifies them. This UUID follows RFC-4122¹ standard.

| | |
|------------|---|
| NFR-DPT-13 | The system shall be able to use only widely accepted cryptographic algorithms and implementations and shall be unable to execute implementations that do not involve some cryptography experts in their creation. |
|------------|---|

| | |
|------------|--|
| NFR-DPT-19 | The system shall be able to use tools that employ open-source or public-domain encryption methods. |
|------------|--|

Involved modules: **e-auction**

The identities of a Hyperledger Fabric blockchain network are based on X.509² digital certificates. Furthermore, the commitments that hide real bids are generated based on HMAC³ commitment scheme, utilizing the pseudo-identity of the prosumer (MSP ID + public key).

| | |
|------------|---|
| NFR-DPT-18 | The system shall be able to generate encryption keys offline and securely store private keys. |
|------------|---|

Involved modules: **e-auction**

The certificates of the participating to the energy trading network organizations are generated by certification authorities and private keys remain to the possession of the organization and are not stored in the system.

| | |
|-----------|---|
| NFR-RL-01 | The System shall be able to perform a required function under stated conditions for a specified time. The specified time should be defined for each action or operation of the system in order for the system to follow these time constraints of operation |
|-----------|---|

Involved modules: **e-auction**

The bidding process of an auction is always conducted under strict time limitation, during which the prosumers can submit their bids to the auctioneer. No bids can be submitted once the time limitation has passed. The same applies also for the bid revealing, as well as for the energy transaction periods.

| | |
|-----------|--|
| NFR-RL-04 | The system shall implement mechanisms to ensure resilience against human errors and interactions within the system |
|-----------|--|

Involved modules: **e-auction**

The system offers different APIs for different roles of organizations in order to properly interact with the chaincodes. Also, there are several controls for different operations, that inform the user with

¹ <https://tools.ietf.org/html/rfc4122>

² <https://en.wikipedia.org/wiki/X.509>

³

[https://en.wikipedia.org/wiki/HMAC#:~:text=In%20cryptography%2C%20an%20HMAC%20\(sometimes,and%20a%20secret%20cryptographic%20key.](https://en.wikipedia.org/wiki/HMAC#:~:text=In%20cryptography%2C%20an%20HMAC%20(sometimes,and%20a%20secret%20cryptographic%20key.)

appropriate messages in case of inappropriate interactions (e.g. trying to bid after bidding deadline has passed).

NFR-RL-05 The system shall be SPoF-safe (single point of failure).

Involved modules: **e-auction**

The distributed ledger, a copy of which is available for all the participating prosumers, minimizes the possibility of a single point of failure.

4. System analysis

The Blockchain-based Energy Trading System is composed of two modules, the e-auction module and the Blockchain-based Intrusion and Anomaly Detection module. Both are based on Hyperledger Fabric framework [1] which offers the ability to build enterprise blockchain solutions taking into account privacy of transactions and permissioned access based on roles that are agreed by the consortium of organizations that form the network. In Table 3, the pros and cons of permissioned and permissionless blockchain networks are presented, while in Table 4 a comparative analysis of some of the most popular blockchain platforms that are used by companies for building blockchain applications is provided.

Table 3: Permissioned vs permissionless blockchain networks

| Category | Permissioned | Permissionless |
|------------------|--|---|
| Speed | High throughput and medium latency | Low throughput and slow latency |
| Privacy | Reading/writing rights can be restricted. | Can be achieved though cryptographic techniques with the cost of lower efficiency |
| Ownership | Managed by a pre-defined group of nodes | Public ownership |
| Decentralization | Partially decentralized | Fully decentralized |
| Cost | Cost-effective | Not so cost-effective |
| Consensus | Practical Byzantine Fault Tolerance (PBFT) protocols, tolerating malicious peers and trusting the majority consensus | Proof of Work or Proof of Stake by miners |

Table 4: Comparative analysis of different blockchain platforms

| | Industry focus | Ledger Type | Consensus algorithm | Smart Contracts |
|----------|--------------------|----------------|---------------------|-----------------|
| Ethereum | Cross-Industry | Permissionless | Proof of Work | Yes |
| Fabric | Cross-Industry | Permissioned | Pluggable | Yes |
| R3 Corda | Financial Services | Permissioned | Pluggable | Yes |

| | | | | |
|-----------|--------------------------|-----------------------|--------------------------------------|-----|
| Ripple | Financial Services | Permissioned | Probabilistic Voting | No |
| Quorum | Cross-Industry | Permissioned | Majority Voting | No |
| Sawtooth | Cross-Industry | Permissioned | Pluggable | Yes |
| Iroha | Cross-Industry | Permissioned | Chain-based Byzantine Fault Tolerant | Yes |
| Openchain | Digital Asset Management | Permissioned | Partitioned Consensus | Yes |
| Stellar | Financial Services | Both public & private | Stellar Consensus Protocol | Yes |
| EOS | Cross-Industry | Permissioned | Delegated Proof-of-Stake | Yes |

A permissioned blockchain is more suitable for the Blockchain-based Energy Trading System, because of the much higher throughput of transactions, which do not require computational heavy calculations that the algorithms of public networks such as Proof of Work usually require to establish trust between completely anonymous entities. Also, the permissioned membership, offers more trust among participants as they do not transact with unknown entities. Some of the outstanding features that differentiate Fabric framework from other distributed ledger technologies that led to its selection for the implementation of the blockchain-based energy trading system are the following:

- Permissioned architecture
- Modularity
- Flexible approach to data privacy with data isolation utilizing “channels” or share private data using “data collections”
- Multi-language smart contract support, Go, Java, Javascript
- Flexible endorsement model for reaching consensus between required organizations
- Pluggable consensus mechanism
- Queryable data (key-based queries and JSON queries)
- Open smart contract model
- Low latency of finality/confirmation

The basic entities of a Fabric blockchain network are the following:

- **Peers:** The peers are the nodes of the blockchain network. In the Energy TradingcSystem’s architecture, each node of the network corresponds to a prosumer. Peers are a basic component of the network, because they host ledgers and smart contracts, which will be explained consequently.
- **Ordering service:** Hyperledger Fabric features a node, called an “orderer” which performs the accepted transactions ordering, and, along with other orderer nodes, forms an ordering service. The ordering service creates blocks of transactions which are distributed to all peers for validation. The Fabric’s design is based on deterministic consensus algorithms, thus all the blocks validated by the peers are definitely final and correct.

- **Ledger:** The ledger contains historic data regarding the results of all the e-auctions that have been conducted in the Energy Trading Framework.
- **Membership Service Provider (MSP):** Each node is assigned with two keys, a public and a private. The pairing of these two keys is used to prove the peer's identity. The private key is used to produce a digital signature on a transaction. The MSP contains the corresponding public key and it is used to verify that the signature attached to a transaction is valid. The MSP mechanism ensures that the identity of a peer is trusted and recognized by the rest of the network, without revealing the corresponding private key.
- **Chaincodes:** Also referred as smart contracts. Chaincode defines all the rules under which all the transactions of the Energy Trading Framework must be conducted. For example, it contains algorithms that determine the outcome of the e-auctions and the incentives mechanism.

4.1 Component model

4.1.1 E-auction module

The participants of the developed energy trading platform are prosumers, meaning entities which both consume and produce energy, and ESCOs which act as token account administrators for the prosumers. Each prosumer can apply to the ESCO to obtain a certain amount of initial tokens, which will be the same for all the participants. These tokens will be used by the stakeholders in order to conduct the financial transactions in the context of the energy trading. The token accounts are handled by the corresponding chaincode, which will be further analyzed later in the document. The prosumers are equipped with Photovoltaic panels (PV) and Energy Storage Systems (ESS). Software agents are assigned to each prosumer, receiving input from prosumer's infrastructure tools, which perform 15-minute PV generation and electric energy consumption forecast. Even though those tools are not considered part of the Energy Trading System, they are essential because they provide the input data for the determination of the role of the prosumer to the market and the corresponding participation parameters (auction parameters for the auctioneers and bid parameters for the bidders). Usually, those tools are based on machine learning forecasting algorithms which based on historic data timeseries and weather forecasting data, perform short-term production and consumption forecasting.

The communication is performed through a channel of a private Fabric blockchain network. An example network with 2 prosumers is illustrated in Figure 2.

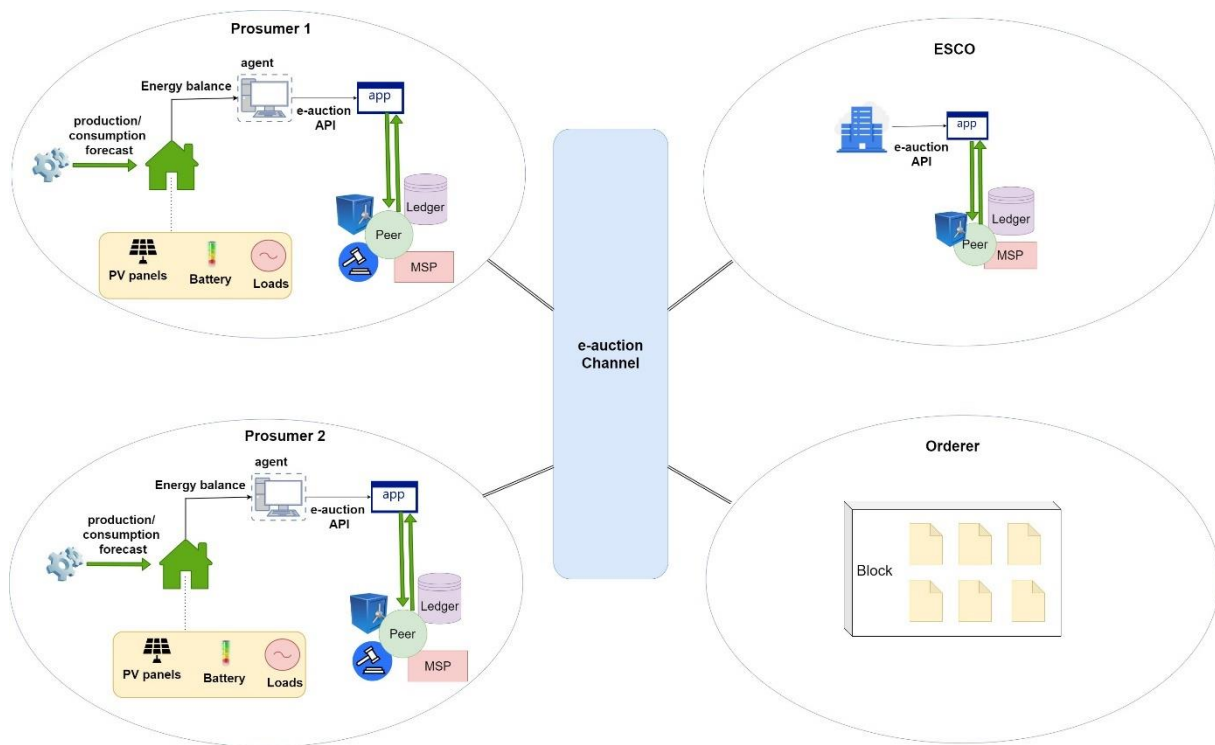


Figure 2: e-auction Fabric network architecture

4.1.1.1 E-auction mechanism

The energy transactions are based on a sealed bid auction mechanism. The auctions take place periodically, in certain time slots. The prosumers are always aware whether they will have a surplus or a deficit of energy in the next time slot. If they will have a surplus, they will become the auctioneers, conducting their own auction, and selling their extra energy. If they will have a deficit of energy, then they will take part in the auctions, as bidders, trying to acquire the energy they will need.

The auction mechanism used is the Vickrey-Clarke-Groves (VCG) auction [2], which is a slight differentiation of the traditional Vickrey Auction. The VCG mechanism is a type of sealed-bid auction of multiple items (in our case kilowatt hours) and leads to multiple winners. Each bidder submits a bid that represents his/her valuation of the items, without knowing the bids of the other bidders. So, the auction mechanism gives the bidders an incentive to bid their true valuations, since they are not aware of the bids submitted by the rest of the bidders. The VCG auction is suitable for our case, because the total energy the bidders request may not be equal to the energy that the auctioneer is selling. Each bidder needs a certain amount of energy to satisfy his/her needs, no more or no less than that. Thus, the energy that is available for sale needs to be split in the portions that the bidders request.

At each auction, the auctioneer declares the amount of energy he/she is willing to sell, which is his/her surplus of energy. The bidders declare the amount of energy they are willing to buy and their bid in tokens. The determination of the winners is based on the following logic: The total energy sold to the bidders needs to be less or equal to the energy that the auctioneer sells, while the total value of the winning bids (which is the total value of the tokens the auctioneer eventually receives) is maximized. An example follows to demonstrate the function of the VCG auction mechanism:

```
VCG auction
Total amount of energy available for sale: 200 kWhs
Participating Bidders:
Bidder 1 [bid = 10 virtual coins, energy request = 50 kWhs]
Bidder 2 [bid = 100 virtual coins, energy request = 20 kWhs]
Bidder 3 [bid = 50 virtual coins, energy request = 20 kWhs]
Bidder 4 [bid = 150 virtual coins, energy request = 200 kWhs]
Bidder 5 [bid = 150 virtual coins, energy request = 250 kWhs]

Auction winners
Total value of the winning bids:160
Total energy sold:90
Winning Bidders:
Bidder 3 [bid = 50 virtual coins, energy request = 20 kWhs]
Bidder 2 [bid = 100 virtual coins, energy request = 20 kWhs]
Bidder 1 [bid = 10 virtual coins, energy request = 50 kWhs]
```

Figure 3: VCG auction mechanism demonstration with 3 winners

Here, the winners are the bidders 1,2 and 3, since the total value of their bids is the maximum possible and the sum of the energy they requested is less than the 200 kWh that are available for sale. Bidder number 5 obviously cannot be the winner of the auction, since his request of 250 kWh exceeds the amount of energy which is available for sale. The energy that was eventually not sold, will be stored back to the auctioneer's battery for future use.

If, for example, bidder number 4 offered 170 tokens for the same energy request, he/she would be the winner, since the tokens the auctioneer will receive are maximized:

```
VCG auction
Total amount of energy available for sale: 200 kWhs
Participating Bidders:
Bidder 1 [bid = 10 virtual coins, energy request = 50 kWhs]
Bidder 2 [bid = 100 virtual coins, energy request = 20 kWhs]
Bidder 3 [bid = 50 virtual coins, energy request = 20 kWhs]
Bidder 4 [bid = 170 virtual coins, energy request = 200 kWhs]
Bidder 5 [bid = 150 virtual coins, energy request = 250 kWhs]

Auction winners
Total value of the winning bids:170
Total energy sold:200
Winning Bidders:
Bidder 4 [bid = 170 virtual coins, energy request = 200 kWhs]
```

Figure 4: VCG auction mechanism demonstration with 1 winner

It must be noted that in the case that bidder 4 also offered 160 tokens, then the winners would still be the bidders 1,2 and 3 because the algorithm chooses the first bidders to maximize the value of the winning bids. This will be further explained later, with the demonstration of the incentive mechanism.

```
VCG auction
Total amount of energy available for sale: 200 kWhs
Participating Bidders:
Bidder 1 [bid = 10 virtual coins, energy request = 50 kWhs]
Bidder 2 [bid = 100 virtual coins, energy request = 20 kWhs]
Bidder 3 [bid = 50 virtual coins, energy request = 20 kWhs]
Bidder 4 [bid = 160 virtual coins, energy request = 200 kWhs]
Bidder 5 [bid = 150 virtual coins, energy request = 250 kWhs]

Auction winners
Total value of the winning bids:160
Total energy sold:90
Winning Bidders:
Bidder 3 [bid = 50 virtual coins, energy request = 20 kWhs]
Bidder 2 [bid = 100 virtual coins, energy request = 20 kWhs]
Bidder 1 [bid = 10 virtual coins, energy request = 50 kWhs]
```

Figure 5: VCG auction mechanism demonstration, winners defined by priority of bidding

There are certain strategies which determine:

- 1) In the case of an auctioneer, the accepted price per kWh. The steps are the following:
 - The auctioneer calculates his/her surplus of energy
 - Calculates how much it cost him/her to produce it using the Levelized Cost of Energy (LCOE) factor
 - There is a pre-determined percentage of profit he/she wants to make
 - Calculates the minimum price per kWh he/she would accept in order to make that profit

This way, a prosumer who sells energy will either sell his/her stored energy at a profitable price or consume his/her own energy.

- 2) In the case of a bidder, the bid he/she is going to submit for the energy needed. The steps are the following:
 - The bidder calculates his/her deficit of energy
 - Calculates how much it would cost to produce it using the LCOE
 - There is a pre-determined range of percentages he/she want to save by participating in the auction (0-15%)
 - Calculates the optimal price for him/her and submits it as a bid

This way, a prosumer buys energy in a price which ensures that he/she will save coins from the transaction.

These strategies are compatible with the nature of the VCG auction, since they ensure that each participant of the e-auction will submit a bid that reflects his/her true valuation of the energy. Before the e-auction is initialized, the auctioneer declares the minimum price/kWh which is accepted (as calculated according to his/her benefit). The bidders interested already know how much energy they need and the price they are willing to offer. If their price is within the restraints of the auctioneer, they participate to the auction. Otherwise they will participate in the next auction conducted at the current time slot.

To summarize the operation of the auction mechanism:

1. Prosumer's security status is checked by the Intrusion Detection System. Prosumers with security issues are not allowed to participate in the market, until the issues are fixed and the severity level of the risk assessment regarding their devices is back to acceptable values.
2. Prosumers are aware whether they have a surplus of energy or not (calculated based on input from consumption/production forecasting tools).
3. Prosumers with surplus of energy declare that they will conduct e-auctions. The e-auctions are conducted based on the priority table. The prosumers with a deficit of energy declare that they will participate at the first e-auction.
4. The auctioneer informs the potential participants of the e-auction about the minimum accepted price per kWh.
5. The potential participants are aware of the energy they need and calculate the price per kWh they offer (calculated based on the LCOE factor). The bid they submit is also restricted by the total amount of tokens they own, meaning they won't be allowed to submit a bid which leads them to bankruptcy (restriction calculated based on input from the bidder's token account).
6. The prosumers who don't meet the auctioneer's requirements do not participate in the e-auction and wait until the next e-auction is initiated.
7. The participants of the e-auction submit their sealed bids and the amount of energy they are willing to purchase, during a predefined time period.
8. After the bidding deadline, the bidders reveal their bids, again during a predefined time period.
9. When the revealing deadline has passed and all bids have been revealed, the auctioneer invokes the chaincode implementing the e-auction mechanism which runs the e-auction algorithm and the winners are decided and announced to OTSC tool.
10. The OTSC tool of EMO module receives the transactions that derive from the e-auction results from the auctioneer and decides upon confirmation/rejection for each energy transaction. The energy involved with rejected transactions remains stored in the auctioneers Energy Storage System.
11. The auctioneer updates the results and the chaincode announces the final results to the participants.
12. The transfer of energy from the auctioneer to the winners is checked through comparison of the measurements from the smart meters monitoring the operation of both the buying and the selling stakeholders.
13. After the confirmation of energy transactions, the winners invoke the chaincode to transfer tokens to the winner's account.

Steps 3 through 8 are repeated for the next auctioneer in the priority order until all the auctioneers conduct their e-auctions. All prosumers with a surplus of energy will initiate their e-auctions, even if, eventually, there will be no prosumers participating. The energy transactions at each time slot refer to energy that will be traded in the next time slot. This is why the consumption/production forecasting functionality is necessary. At each time slot, there will be multiple e-auctions taking place, always one at a time. The e-auctions will be conducted based on the priority table.

At the first e-auction of a time slot, the possible scenarios for each bidder who participates are three:

1. He/she is among the winners, receives the agreed amount of energy at a predefined time during the next timeslot and pays the agreed price in tokens.
2. He/she is not among the winners.
3. He/she is among the winners, but the OTSC tool of EMO rejects the transaction due to microgrid stability issues.

The bidders who experience scenarios 2 or 3, will participate in the e-auction by the prosumer who is next in the priority table.

The priority order in which the prosumers conduct their e-auctions is a form of incentive, in the sense that someone is rewarded for participating in the energy trading process, either by selling or by buying. The reward is that, as auctioneers, they conduct their e-auctions earlier than the rest of the auctioneers, maximizing the possibilities to sell the energy they have available.

As mentioned earlier, the e-auction's algorithm always chooses the first bidders to maximize the value of the winning bids. Thus, the priority table is an incentive for the bidders too. The order in which they will participate in the e-auction will be determined by the priority table, this time excluding the auctioneers. The higher someone is at the priority table, the earlier he/she will participate in the e-auction. At the above example, bidder number 4 was lower at the priority table than 1,2 and 3.

4.1.1.2 Penalty mechanism

All the actions performed by the prosumers, in the context of the energy market, are fully automated, as they are determined by the algorithms run by the corresponding agents and the chaincode. Therefore, at this point of the development, there is no need for a penalty mechanism, since the consistency of the prosumers is guaranteed. Although, in future extensions of the Energy Trading System, users will be granted the capability of handling the prosumer's actions in the energy market, and the outcome of the algorithms run by the agents will be used as suggestions to the user. Thus, a penalty mechanism will be necessary because the consistency of the users cannot be guaranteed in this case. This subsection presents the penalty mechanism that will be implemented in the future extension of the Energy Trading System.

As far as the priority table is concerned, the prosumers are sorted based on their energy contribution to the market (sum of the energy they have sold and bought throughout the whole operation of the developed market mechanism and until, but not including, the current timeslot). The first prosumer of the table is the first to conduct an e-auction at a specific time slot, the second conducts his/her e-auction second and so on. The table contains all the prosumers, no matter if they are buyers or sellers, because these roles may and will change through the course of the time slots.

Table 5: Sample priority table

| PROSUMERS | ENERGY CONTRIBUTION(kWh) | TEMP |
|-----------|--------------------------|------|
| 5 | 2000 | 0 |
| 1 | 1800 | 0 |
| 3 | 1200 | 0 |
| 6 | 1000 | 0 |
| 4 | 750 | 0 |
| 2 | 230 | 0 |

If a prosumer receives a penalty, then the amount of tokens he/she has to pay replaces his/her Energy Contribution (EC) at the table, in the form of a negative number, and his/her EC at that time slot is transferred to the Temp column. The cases in which a prosumer is imposed to a penalty will be analyzed later.

For example, if prosumer number 1 has to pay 50 tokens as a penalty, then the table will look like this:

Table 6: Sample priority table with prosumer 1 being imposed 50 tokens of penalty

| PROSUMERS | ENERGY CONTRIBUTION (kWh) | TEMP |
|-----------|---------------------------|------|
| 5 | 2000 | 0 |
| 3 | 1200 | 0 |
| 6 | 1000 | 0 |
| 4 | 750 | 0 |
| 2 | 230 | 0 |
| 1 | -50 | 1800 |

The table is always sorted based on the EC column, so obviously, prosumer number 1 will end up at the last place. If, for example, prosumer number 4 has to pay 100 tokens as a penalty, then the table will look like this:

Table 7: Sample priority table with prosumer 1 being imposed 50 tokens of penalty and prosumer 4 being imposed 100 tokens of penalty

| PROSUMERS | ENERGY CONTRIBUTION (kWh) | TEMP |
|-----------|---------------------------|------|
| 5 | 2000 | 0 |
| 3 | 1200 | 0 |
| 6 | 1000 | 0 |
| 2 | 230 | 0 |
| 1 | -50 | 1800 |
| 4 | -100 | 750 |

This will result in prosumers number 1 and 4 being last at the order in which they conduct and participate in the e-auctions, as a form of punishment. The prosumers will pay their penalty at any time slot they can afford it, and once this is done, they will take their place back at the sorting, and their Temp column will be zero again. They are not obligated to pay the whole penalty. They can pay it in doses through the course of the time slots. In this case the table will be sorted accordingly. For example, prosumer number 1 pays the penalty of 50 coins:

Table 8: Sample priority table with prosumer 1 having paid 50 tokens of penalty and prosumer 4 being imposed 100 tokens of penalty

| PROSUMERS | ENERGY CONTRIBUTION (kWh) | TEMP |
|-----------|---------------------------|------|
| 5 | 2000 | 0 |
| 1 | 1800 | 0 |
| 3 | 1200 | 0 |
| 6 | 1000 | 0 |

| | | |
|---|------|-----|
| 2 | 230 | 0 |
| 4 | -100 | 750 |

For simplicity, we assume that the EC of the other prosumers does not change through the course of different time slots.

A prosumer may be imposed to a penalty in two cases:

- As an auctioneer, if the energy promised is not delivered to a winning bidder. This confirmation will occur from the comparison of the measurements from the smart meters monitoring the operation of the selling and the buying prosumer.
- As a participant of the e-auction, if the agreed payment is not delivered to the auctioneer's token account.

In the first case, the auctioneer has to return the payment to the winner plus a percentage of it as a penalty. In the second case, the bidder has to return the agreed payment to the auctioneer plus a percentage as a penalty.

If a prosumer has a penalty pending, then he/she will be able to participate in an e-auction as a bidder normally or conduct his/her own e-auction. He/she will be excluded only from the e-auctions conducted by the prosumer he/she owns to. If he/she wants to participate, he/she will have to pay his/her penalty to him/her. The energy he/she buys or sells during his "restriction" time, will be updated in the temp column.

If a prosumer owns a penalty:

- The prosumer is informed by the forecasting tool about the production/consumption balance in the next time slot.
- If there will be a surplus of energy, meaning he/she will conduct an e-auction in the next time slot, he/she pays as much of the penalty as possible, with the restriction that he will still have a 20% of his initial coins after the payment. If the penalty is not entirely covered, this process will be repeated in the next time slots, until the whole penalty is paid.
- If there is a deficit of energy, the prosumer calculates his/her optimal bid as explained before, and with the coins remaining, he/she pays as much of the penalty as possible, with the same restrictions as with the case of the prosumer with a surplus of energy.

In the case that, after the course of several time slots the penalty is not yet paid to its entity, the bank takes over and pays the rest of the penalty to the prosumer who is owned to, and the rest of the debt is transferred to the bank. The prosumer who owns the penalty pays the rest of it to the bank in the same way in which he/she pays it to other prosumers, plus a percentage in order to ensure a profit for the bank.

It must be noted that, in the current, fully automated version of the Energy Trading System, the prosumers will be sorted only according to their contribution to the market, without taking under consideration the Temp column, since the penalty mechanism is not included.

4.1.1.3 Vickrey-Clarke-Groves chaincode

Table 9 presents the Vickrey Clarke-Groves chaincode functions, along with their request type (query or invoke) the input parameters and their description.

Table 9: Vickrey-Clarke-Groves chaincode functions

| Name | Request type | Input(JSON) | | Description |
|------------|--------------|----------------------|-----------|---|
| | | Param | Type | |
| initialize | invoke | Auctioneer | struct | The Auctioneer parameter is a struct representing the peer node that initializes the e-auction. |
| | | Product | struct | The Product parameter is a struct with the details of the product that is being sold. |
| | | ReservePrice | float | A deposit of tokens is withheld from the bidders when the submit a bid. |
| | | InitTimestamp | timestamp | The timestamp of the initialization of the e-auction. |
| | | BiddingDeadline | timestamp | The timestamp indicating the bidding period deadline. |
| | | RevealingDeadline | timestamp | The timestamp indicating the revealing bid phase deadline. |
| | | TransactionTimestamp | timestamp | The timestamp indicating when the energy transaction is going to take place. |
| | | AuctionDeadline | timestamp | The timestamp indicating the auction deadline |
| | | AwardingDeadline | timestamp | The timestamp indicating the awarding phase deadline. |
| | | MarketID | string | The unique identifier of the market to which the auction belongs. |
| | | PriorityTableID | string | The unique identifier of the priority table that keeps the contributions to the market for each participant organization. |

| | | | | |
|-------------|--------|------------|---------|--|
| get | query | AuctionNum | string | Every time an e-auction is initialized, a UUID is registered for that auction which is the auctionNum parameter. |
| getAuctions | query | - | - | - |
| submit | invoke | AuctionNum | string | Parameter already described |
| | | Bid | float | The 'Bid' parameter represents the number of tokens that the bidder offers. |
| | | Amount | integer | The 'Amount' parameter represents the amount of energy in kWh that the bidder bids for. |
| reveal | invoke | AuctionNum | string | Parameter already described. |
| | | Bid | float | Parameter already described. |
| | | Amount | integer | Parameter already described. |
| award | invoke | AuctionNum | string | Parameter already described. |
| settle | invoke | AuctionNum | string | Parameter already described. |

Tables 10-15 describe the structures that the chaincode manages:

Table 10: Vickrey Clarke-Groves auction structure

| Auction | | |
|---------------------|--------|---|
| Name | Type | Description |
| Auctioneer | struct | Auctioneer struct is composed of the MSP ID and the Cert ID of the blockchain node that uniquely identifies it within the network. |
| Product | struct | Product is a struct composed of the name of the product (string), the units of the measurement (string) and the amount of product that is being sold (int). |
| ReservePrice | float | The reserve price indicates the minimum amount of tokens/KWh that is accepted as a bid. |

| | | |
|-----------------------------|------------------|---|
| HiddenBids | slice of structs | This is a slice including HiddenBid structs, equal to the number of submitted bids. |
| RevealedBids | slice of structs | This is a slice including RevealedBid structs, equal to the number of revealed bids. |
| InitTimestamp | timestamp | The timestamp of the initialization of the e-auction. |
| BiddingDeadline | timestamp | The timestamp indicating the bidding period deadline. |
| RevealingDeadline | timestamp | The timestamp indicating the revealing bid period deadline. |
| TransactionTimestamp | timestamp | The timestamp indicating when the energy transaction is going to take place. |
| AwardingDeadline | timestamp | The timestamp indicating the awarding phase deadline |
| AuctionDeadline | timestamp | The timestamp indicating the auction deadline. |
| Results | struct | 'Results' parameter represents a struct that includes the results of the e-auction. |
| AuctionNum | string | Every time an e-auction is initialized, a UUID is registered for that auction which is the auctionNum parameter. |
| MarketID | string | The unique identifier of the market to which the auction belongs. |
| PriorityTableID | string | The unique identifier of the priority table that keeps the contributions to the market for each participant organization. |
| State | string | The 'State' parameter is a string indicating the step of an e-auction process and can take the following values: |

| | | |
|--|--|---|
| | | 'initialized', 'bidding', 'revealing', 'refunding', 'settling' |
|--|--|---|

Table 11: Participant structure

| Participant | | |
|-------------|--------|--|
| Name | Type | Description |
| MspID | string | MspID indicates the ID of the Membership Service Provider of the participant organization. |
| CertID | string | CertID is the public key of the peer |

Table 12: Commodity structure

| Commodity | | |
|-----------|---------|--|
| Name | Type | Description |
| Name | string | The name of the product for sale. |
| Units | string | The units of measurement for the product for sale. |
| Amount | integer | The amount of the product for sale specified in measurement units as indicated by 'Units' parameter. |

Table 13: Result structure

| Result | | |
|-----------------|------------------|---|
| Name | Type | Description |
| Winners | slice of structs | The slice 'Winners' includes all the winning bidders that are represented with 'Participant' structs. |
| RejectedWinners | slice of structs | The slice 'RejectedWinners' includes all the winning bidders whose energy transactions are |

| | | |
|----------------------|----------------|--|
| | | rejected by the OTSC EMO tool., due to grid destabilization issues. |
| ClearingPrice | float | The total amount of tokens from all the winning bids. |
| EnergySold | integer | The amount of energy that was sold from all the winning bids. |
| EnergyNotSold | integer | The difference between the amount of energy that was for sale and the amount of energy that was eventually sold. |
| Payments | slice of float | The slice 'Payments' includes the payment amount in tokens for each winner. |
| Shares | Slice of int | The slice 'Shares' includes the amount of energy that is assigned to each winner. |

Table 14: HiddenBid structure

| HiddenBid | | |
|------------------|---------------|---|
| Name | Type | Description |
| Bidder | struct | The bidder is represented by a struct of type 'Participant' |
| HiddenBid | slice of byte | The 'HiddenBid' parameter is the encrypted representation of the actual bid (tokens+amount of energy) |
| Nonce | slice of byte | The 'Nonce' parameter is a cryptographic key that is produced to encrypt the bid. |

Table 15: RevealedBid structure

| RevealedBid | | |
|-------------|------|-------------|
| Name | Type | Description |

| | | |
|---------------|---------|---|
| Bidder | struct | The bidder is represented by a struct of type 'Participant' |
| Bid | float32 | The 'Bid' parameter represents the revealed bid token amount that was included in the HiddenBid. |
| Amount | int | The 'Amount' parameter represents the revealed amount of energy that was included in the HiddenBid. |

4.1.1.4 ERC20 token chaincode

Table 16 presents the ERC20 chaincode functions, along with their request type (query or invoke) the input parameters and their description.

Table 16: ERC20 token chaincode functions

| Name | Request type | Input | | Description |
|----------------------|--------------|--------------|--------|---|
| | | Param | Type | |
| initToken | invoke | Name | string | The name of the token that is initialized. |
| | | Symbol | string | The string representation of the token that is initialized. |
| | | TotalSupply | float | The total token units that are available for the token that is initialized. |
| getOwnBalance | query | Name | string | Parameter already described. |
| | | Symbol | string | Parameter already described. |
| | | BalanceOwner | struct | The account balance of the peer node that is querying the blockchain. |
| update | invoke | Name | string | Parameter already described. |
| | | Symbol | string | Parameter already described. |
| | | Amount | int | The updated account balance of a peer node's account. |
| | | Account | struct | The 'Account' parameter is a struct of type 'Participant' that indicates the peer node whose account should be updated. |
| transfer | invoke | Name | string | Parameter already described. |

| | | | | |
|--------------------------|--------|--------|-----------------|---|
| | | Symbol | string | Parameter already described. |
| | | Amount | int | The number of tokens to be transferred from one account to another. |
| | | From | struct | The 'From' parameter is a struct of type 'Participant' that indicates the peer node's account from which the tokens will be transferred. |
| | | To | struct | The 'To' parameter is a struct of type 'Participant' that indicates the peer node's account to which the tokens will be transferred. |
| withhold | invoke | Name | string | Parameter already described. |
| | | Symbol | string | Parameter already described. |
| | | Amount | int | The number of tokens to be withheld from the account. |
| | | From | struct | The 'From' parameter is a struct of type 'Participant' that indicates the peer node's account from which the tokens will be withheld. |
| release | invoke | Name | string | Parameter already described. |
| | | Symbol | string | Parameter already described. |
| | | Amount | int | The number of tokens to be released to the accounts specified by 'To' parameter. |
| | | To | Slice of struct | The 'To' parameter is a slice including structs of type 'Participant' that indicate the peer nodes' accounts to which the withheld number of tokens will be released. |
| apply | invoke | Name | string | Parameter already described. |
| | | Symbol | string | Parameter already described. |
| getApplicantsList | query | Name | string | Parameter already described. |
| | | Symbol | string | Parameter already described. |

Table 17 describes the structure that the chaincode manages:

Table 17: Token structure

| Token | | |
|------------------------|--------------------|---|
| Name | Type | Description |
| Name | string | The name of the token. |
| Symbol | slice of byte | The string representation of the token's symbol. |
| TotalSupply | float | The total token units that are available for the token. |
| AccountBalances | Map[string, float] | A map with key the concatenation of MspID and CertID strings of a peer node owning an account and value the account's token balance. |
| TokenApplicants | slice of struct | The token applicants slice includes a list with structs of type 'Participant' that represent the peer nodes that have applied for an account. |

4.1.1.5 Priority table chaincode

Table 18 presents the priority table chaincode functions, along with their request type (query or invoke) the input parameters and their description.

Table 18: Priority table chaincode functions

| Name | Request type | Input | | Description |
|------------------------------|--------------|-------------|--------|--|
| | | Param | Type | |
| initPriorityTable | invoke | - | - | - |
| getPriorityTable | query | ID | string | The unique identifier of the priority table. |
| getPriorityTables | query | - | - | - |
| getMarketContribution | query | ID | string | Parameter already described. |
| | | Participant | struct | The 'Participant' parameter is a struct of type Participant and represents the identification information of |

| | | | | |
|--------------------------------------|--------|-------------|--------|--|
| | | | | the Organization's peer that queries the blockchain. |
| addParticipantToPriorityTable | invoke | ID | string | Parameter already described. |
| | | Participant | struct | The 'Participant' parameter is a struct of type Participant and represents the identification information of the Organization's peer that should be added to the priority table. |
| updateMarketContribution | invoke | ID | string | Parameter already described. |
| | | Participant | struct | The 'Participant' parameter is a struct of type Participant and represents the identification information of the Organization's peer whose contribution to the market should be updated. |
| | | Amount | int | The amount of energy that should be added to the Participant's contribution to the market. |

Table 19 describes the structure that the chaincode manages:

Table 19: Priority Table structure

| Priority Table | | |
|---------------------------|------------------|---|
| Name | Type | Description |
| ID | string | The unique identifier of the priority table. |
| Participants | slice of structs | Includes the list of the Organizations' peers that are registered to the priority table. |
| MarketContribution | Map[string, int] | A map with key the concatenation of MspID and CertID strings of an organization's peer node and value the contribution of the organization to the market. |

4.1.1.6 Market chaincode

Table 20 presents the priority table chaincode functions, along with their request type (query or invoke) the input parameters and their description.

Table 20: Market chaincode functions

| Name | Request type | Input | | Description |
|-----------------------|--------------|---------------------|-----------|---|
| | | Param | Type | |
| initMarket | invoke | ApplicationDeadline | timestamp | The timestamp representing the deadline of the applying phase for the market. |
| getMarket | query | ID | string | The unique identifier of the market. |
| getMarkets | query | - | - | - |
| apply | invoke | ID | string | Parameter already described. |
| | | PriorityTableID | string | The unique identifier of a priority table, that includes the energy contributions to the market. |
| | | Role | string | This parameter takes 2 values, either 'Bidder' or 'Auctioneer' and represents the role of the applicant to the market. |
| removeFromList | invoke | ID | string | Parameter already described. |
| | | Role | string | Parameter already described. |
| | | Participant | struct | The 'Participant' parameter is a struct of type Participant and represents the identification information of the Organization's peer that |

| | | | | |
|--|--|--|--|--|
| | | | | will be removed from the list of Bidders or from the list of Auctioneers depending on the value of 'Role' parameter. |
|--|--|--|--|--|

Table 21 describes the structure that the chaincode manages:

Table 21: Market structure

| Market | | |
|----------------------------|------------------|--|
| Name | Type | Description |
| ID | string | The unique identifier of the market. |
| Participants | slice of structs | Includes the list of the Organizations' peers that are registered to the market. |
| Bidders | slice of structs | Includes the list of the Organizations' peers that are registered to the market with Bidder role. |
| Auctioneers | slice of structs | Includes the list of the Organizations' peers that are registered to the market with Auctioneer role. |
| Auctions | Slice of strings | Includes the list of the auction unique identifiers that are part of a market. |
| ApplicationDeadline | timestamp | The timestamp indicating the deadline of the application phase for a market. After this deadline passes no more participants are able to register to the market. |

4.1.1.7 E-auction module APIs

For the interaction with the blockchain, the e-auction module offers REST APIs for client applications. Depending on the role of the organization within the blockchain network, 2 different APIs provide

access to the appropriate chaincode functions. The first one is for organizations that are auctioneers or bidders and the second one is for ESCO organizations that are administrators of the network's token and are responsible for managing the account balances according to the e-auction financial transactions. The definition of the APIs is presented in sections 4.1.1.7.1 and 4.1.1.7.2, according to OpenAPI 3.0 specification format.

Auctioneer-Bidder API

The auctioneer-bidder API enables the organizations to interact both with the ERC20 chaincode and the Vickrey-Clarke-Groves chaincode. Through the available endpoints, they are able to apply for a token account if they do not already have one, retrieve their own balance at any time and transfer tokens from their account to another, in order to complete the financial settlements that derive from e-auction results.

| | | |
|--|-----------------------|--|
| ERC20 token Token account processes | | ▼ |
| POST | /apply | Apply for token account |
| GET | /getBalance | Each organization can retrieve its own balance |
| POST | /transfer | Transfer tokens to another organization's account |
| Vickrey Clarke Groves auction Vickrey Clarke-Groves auction lifecycle processes | | ▼ |
| POST | /initVickreyCGAuction | Initialize a new Vickrey auction |
| GET | /getVickreyCGAuction | Returns Vickrey auction state |
| POST | /VCgsubmitSealedBid | Submit commitment of bid to a Vickrey auction |
| POST | /VCgreveal | Bidders reveal their bids after the bidding deadline passes |
| POST | /VCgaward | Auctioneer invokes the chaincode function that implements algorithm to award winners |
| POST | /VCgsettle | Winners transfer tokens corresponding to their bid to the Auctioneer's account |
| Market Market retrieval and registration processes | | ▼ |
| GET | /getMarkets | Returns list of Markets |
| POST | /applyToMarket | Apply to an existing market as Auctioneer or Bidder |
| Priority Table Priority table retrieval | | ▼ |
| GET | /getPriorityTable | Returns list of Priority Tables |

Figure 6: Auctioneer-Bidder API endpoints

POST

/apply Apply for token account

Parameters

Try it out

No parameters

Request body required

application/json

Name and symbol of the token for which the application is done

Example Value

Schema

```
{
  "Name": "string",
  "Symbol": "string"
}
```

Responses

| Code | Description | Links |
|------|--|----------|
| 200 | <div>successful operation</div> <div>Media type</div> <div>text/plain</div> <div>Controls Accept header.</div> <div> <div>Example Value</div> <div>Schema</div> </div> <div>string</div> | No links |
| 400 | <div>Bad request</div> <div>Media type</div> | No links |
| 500 | <div>Unable to invoke apply in ERC20 chaincode</div> <div>Media type</div> | No links |

Figure 7: /apply request parameters and responses

GET
/getBalance
Each organization can retrieve its own balance

Parameters
Try it out

| Name | Description |
|---|--|
| name * required string (query) | Name of the token <input type="text" value="name - Name of the token"/> |
| symbol * required string (query) | String representation of the token symbol <input type="text" value="symbol - String representation of the token symbol"/> |
| mspID * required string (query) | Identifier of the organization's MSP <input type="text" value="mspID - Identifier of the organization's MSP"/> |
| certID * required string (query) | Identifier of the organization's certificate <input type="text" value="certID - Identifier of the organization's certificate"/> |

Responses

| Code | Description | Links |
|------|--|----------|
| 200 | successful operation Media type <input type="text" value="application/json"/> Controls: Accept: header Example Value Schema <div> { "Balance": "string" } </div> | No links |
| 400 | Bad request. One or more parameters missing | No links |
| 500 | Unable to query the blockchain | No links |

Figure 8: /getBalance request parameters and responses

POST

/transfer

Transfer tokens to another organization's account

Parameters

Try it out

No parameters

Request body

required

application/json

Name and symbol of the token, amount of token units, identities of sender and receiver

Example Value | Schema

```
{
  "Name": "string",
  "Symbol": "string",
  "Amount": 0,
  "From": {
    "mspID": "string",
    "certID": "string"
  },
  "To": {
    "mspID": "string",
    "certID": "string"
  }
}
```

Responses

| Code | Description | Links |
|------|---|----------|
| 200 | successful operation | No links |
| | <div>Media type</div> <div>text/plain</div> <div>Controls: Accept: header</div> <div>Example Value Schema</div> <div>string</div> | |
| 400 | Bad request | No links |
| | Media type | |
| 500 | Unable to invoke transfer in ERC20 chaincode | No links |
| | Media type | |

Figure 9: /transfer request parameters and responses

POST /initVickreyCGAuction Initialize a new Vickrey auction

[Try it out](#)

Parameters

No parameters

Request body required application/json

Parameters needed to initialize a new Vickrey auction

[Example Value](#) | [Schema](#)

```
{
  "Auctioneer": {
    "mspID": "string",
    "certID": "string"
  },
  "Product": {
    "Name": "string",
    "Units": "string",
    "Amount": 0
  },
  "Deposit": 0
}
```

Responses

| Code | Description | Links |
|------|---|----------|
| 200 | successful operation | No links |
| 400 | Bad request | No links |
| 500 | Unable to invoke initialize in VickreyAuction chaincode | No links |

Figure 10: /initVickreyCGAuction request parameters and responses

GET /getVickreyCGAuction Returns Vickrey auction state

[Try it out](#)

Parameters

| Name | Description |
|---|---|
| auctionNum <small>* required</small> | UUID of Vickrey auction |
| string (query) | <input type="text" value="auctionNum - UUID of Vickrey auction"/> |

Responses

| Code | Description | Links |
|------|---|----------|
| 200 | successful operation | No links |
| 400 | Bad request. AuctionNum parameter missing | No links |
| 500 | Unable to query the blockchain | No links |

Figure 11: /getVickreyCGAuction request parameters and responses

POST
/VCGsubmitSealedBid
Submit commitment of bid to a Vickrey auction

Parameters
Try it out

No parameters

Request body required

application/json

Parameters needed to submit bid

Example Value | Schema

```

{
  "AuctionNum": "string",
  "Bid": 0,
  "Amount": 0
}

```

Responses

| Code | Description | Links |
|------|--|----------|
| 200 | successful operation <small>Media type:</small> <div>text/plain</div> <small>Controls Accept header.</small> <div>Example Value Schema</div> <pre> string </pre> | No links |
| 400 | Bad request <small>Media type:</small> | No links |
| 500 | Unable to invoke submit in VickreyAuction chaincode <small>Media type:</small> | No links |

Figure 12: /VCGsubmitSealedBid request parameters and responses

POST

/VCGreveal

Parameters

Try it out

No parameters

Request body required

application/json

Parameters needed to reveal bid

Example Value

Schema

```
{
  "AuctionNum": "string",
  "Bid": 0,
  "Amount": 0
}
```

Responses

| Code | Description | Links |
|------|---|----------|
| 200 | successful operation <div>Media type</div> <div>text/plain</div> <div>Controls Accept header.</div> <div>Example Value</div> <div>Schema</div> <div>string</div> | No links |
| 400 | Bad request <div>Media type</div> | No links |
| 500 | Unable to invoke reveal in VickreyAuction chaincode <div>Media type</div> | No links |

Figure 13: /VCGreveal request parameters and responses

POST

/VCGaward

Parameters

Try it out

No parameters

Request body required

application/json

Parameters needed to award winner

Example Value

Schema

```
{
  "AuctionNum": "string"
}
```

Responses

| Code | Description | Links |
|------|---|----------|
| 200 | successful operation <div>Media type</div> <div>text/plain</div> <div>Controls Accept header.</div> <div>Example Value</div> <div>Schema</div> <div>string</div> | No links |
| 400 | Bad request <div>Media type</div> | No links |
| 500 | Unable to invoke award in VickreyAuction chaincode <div>Media type</div> | No links |

Figure 14: /VCGaward parameters and responses

POST
/VCGsettle

Parameters

Try it out

No parameters

Request body required

application/json

Parameters needed for financial settlement of Vickrey auction

Example Value

Schema

```
{
  "AuctionNum": "string"
}
```

Responses

| Code | Description | Links |
|------|--|----------|
| 200 | successful operation | No links |
| | <div>Media type</div> <div>text/plain</div> <div>Content Accept header</div> <div> <div>Example Value</div> <div>Schema</div> </div> <div>string</div> | |
| 400 | Bad request | No links |
| | Media type | |
| 500 | Unable to invoke settle in VickreyAuction chaincode | No links |
| | Media type | |

Figure 15: /VCGsettle request parameters and responses

Market Market retrieval and registration processes

GET
/getMarkets Returns list of Markets

Parameters

Try it out

No parameters

Responses

| Code | Description | Links |
|------|--|----------|
| 200 | successful operation | No links |
| | <div>Media type</div> <div>text/plain</div> <div>Content Accept header</div> <div> <div>Example Value</div> <div>Schema</div> </div> <div>string</div> | |
| 500 | Unable to query the blockchain | No links |

Figure 16: /getMarkets request responses

POST /applyToMarket Apply to an existing market as Auctioneer or Bidder

Parameters Try it out

No parameters

Request body required application/json

The request body includes the parameters that are needed to apply to a market
 Example Value Schema


```

{
  "ID": "string",
  "PriorityTable": "string",
  "Role": "string"
}
        
```

Responses

| Code | Description | Links |
|------|--|----------|
| 200 | successful operation Media type: text/plain Content Accept Header: Example Value Schema string | No links |
| 400 | Bad request Media type: | No links |
| 500 | Unable to invoke apply in market chaincode Media type: | No links |

Figure 17: /applyToMarket request parameters and responses

Priority Table Priority table retrieval

GET /getPriorityTable Returns list of Priority Tables.

Parameters Try it out

No parameters

Responses

| Code | Description | Links |
|------|--|----------|
| 200 | successful operation Media type: text/plain Content Accept Header: Example Value Schema string | No links |
| 500 | Unable to query the blockchain Media type: | No links |

Figure 18: /getPriorityTable request responses

ESCO API

The ESCO API provides interaction only with the ERC20 chaincode, as the ESCO acts as an administrator for the token accounts and does not take part in energy trading transactions. Through this API, the ESCO is able to initialize a new token, create accounts and initialize account balances, retrieve the list of organizations that apply for account creation, initialize a new market, initialize a priority table that

keeps the energy contribution of each organization and retrieve the list of priority tables (should be only one per channel of organizations).

| ERC20 token <small>Token account processes</small> | | ▼ |
|--|--------------------|--|
| POST | /initToken | Initialize token |
| GET | /getApplicants | Retrieve list of applicant organizations for generating token accounts |
| POST | /updateBalance | Update balance of an organization's account |
| Market | | ▼ |
| POST | /initMarket | Initiate a new market |
| Priority Table | | ▼ |
| POST | /initPriorityTable | Initiate a new priority table to hold the organizations' energy contribution to the market |
| GET | /getPriorityTable | Returns list of Priority Tables. |

Figure 19: ESCO API endpoints

POST
/initToken
Initialize token

Parameters

Try it out

No parameters

Request body required

application/json

Name, symbol and total amount of the token to be initialized

Example Value

Schema

```
{
  "Name": "string",
  "Symbol": "string",
  "TotalSupply": 0
}
```

Responses

| Code | Description | Links |
|------|--|----------|
| 200 | successful operation | No links |
| | <div>Media type</div> <div>text/plain</div> <div>Controls: Accept: header.</div> <div> <div>Example Value</div> <div>Schema</div> </div> <div>string</div> | |
| 400 | Bad request | No links |
| | <div>Media type</div> | |
| 500 | Unable to invoke apply in ERC20 chaincode | No links |
| | <div>Media type</div> | |

Figure 20: /initToken request parameters and responses

GET
/getApplicants
Retrieve list of applicant organizations for generating token accounts

Parameters
Try it out

| Name | Description |
|---|--|
| name * required string (query) | Name of the token <input type="text" value="name - Name of the token"/> |
| symbol * required string (query) | String representation of the token symbol <input type="text" value="symbol - String representation of the token symbol"/> |

Responses

| Code | Description | Links |
|------|--|----------|
| 200 | successful operation Media type: <input type="text" value="text/plain"/> Controls: Accept header. Example Value Schema <div>string</div> | No links |
| 400 | Bad request. One or more parameters missing Media type: | No links |
| 500 | Unable to query the blockchain Media type: | No links |

Figure 21: /getApplicants request parameters and responses

POST

/updateBalance

Update balance of an organization's account

Parameters

Try it out

No parameters

Request body required

application/json

Name and symbol of the token, amount of token units, identities of sender and receiver

Example Value

Schema

```
{
  "Name": "string",
  "Symbol": "string",
  "Amount": 0,
  "From": {
    "mspID": "string",
    "certID": "string"
  }
}
```

Responses

| Code | Description | Links |
|--|--|----------|
| 200 | successful operation | No links |
| <div>Media type</div> <div>text/plain</div> <div>Controls Accept: header</div> <div> <div>Example Value</div> <div>Schema</div> </div> <div>string</div> | | |
| 400 | Bad request | No links |
| <div>Media type</div> | | |
| 500 | Unable to invoke update in ERC20 chaincode | No links |
| <div>Media type</div> | | |

Figure 22: /updateBalance request parameters and responses

| POST /initMarket Initiate a new market | | |
|--|--|----------|
| Parameters | | |
| No parameters | | |
| Responses | | |
| Code | Description | Links |
| 200 | successful operation | No links |
| | <div>Media type</div> <div>text/plain</div> <div>Controls Accept's header</div> <div>Example Value</div> <div>Schema</div> <div>string</div> | |
| 500 | Unable to invoke initMarket in market chaincode | No links |
| | <div>Media type</div> | |

Figure 23: /initMarket request responses

| POST /initPriorityTable Initiate a new priority table to hold the organizations' energy contribution to the market | | |
|--|--|----------|
| Parameters | | |
| No parameters | | |
| Responses | | |
| Code | Description | Links |
| 200 | successful operation | No links |
| | <div>Media type</div> <div>text/plain</div> <div>Controls Accept's header</div> <div>Example Value</div> <div>Schema</div> <div>string</div> | |
| 500 | Unable to invoke initPriorityTable in priorityTable chaincode | No links |
| | <div>Media type</div> | |

Figure 24: /initPriorityTable request responses

4.1.2 Blockchain Based Intrusion and Anomaly Detection module

The Blockchain Based Intrusion and Anomaly Detection system (BIAD) is a peer-to-peer network used to monitor the activity and performance of a set of devices (i.e. smart meter, IEDs, RTUs...), maintaining their status and integrity, and alerting when unusual operation is detected. This is done by detecting changes in log files and in firmware and configuration files in different devices and monitoring the current state of each device in the blockchain system, compiling robust transaction records. Another functionality allows the system administrators to perform periodically on-chain availability-connectivity tests.

The blockchain network is developed using Hyperledger Fabric, an open source project from the Linux Foundation to easily set up a distributed ledger focused on traceability. The network has the following components (See Figure 25):

- Peer Node

For this project case it has been set up only a single peer for a single organization, responsible for all the monitoring interactions with tested devices.

- Orderer Node

It is the central communications node, responsible for maintaining state and consistency of the ledger and mining and distribution of the blocks. There is also a single orderer working for this proof of concept.

- Certificate Authority Server

Manages and provides the access control to the network functionalities through Enrolment Certificates.

- Chaincode

The chaincode is a program which provides logic to the peers, a smart contract, and is necessary to be able to read and update the ledger state and create transactions. In this case, the chaincode is developed in the Go programming language and it is installed on a channel of the blockchain. Its functionalities are ejected in the peers that belong to that channel.

Furthermore, there some other components external to the blockchain network:

- REST API Server

It works as Hyperledger Fabric wallet, it is supported by Fabric SDK and it can be used to create, sign and send transactions to the ledger. In this case, it works as another service, but it can be integrated with the own agent. Provides authentication using token-based authentication.

- Explorer Server

This component is a daemon which sets up a web server to visualize the performance dashboard of the ledger. It is using Hyperledger Explorer, another project from the Linux Foundation.

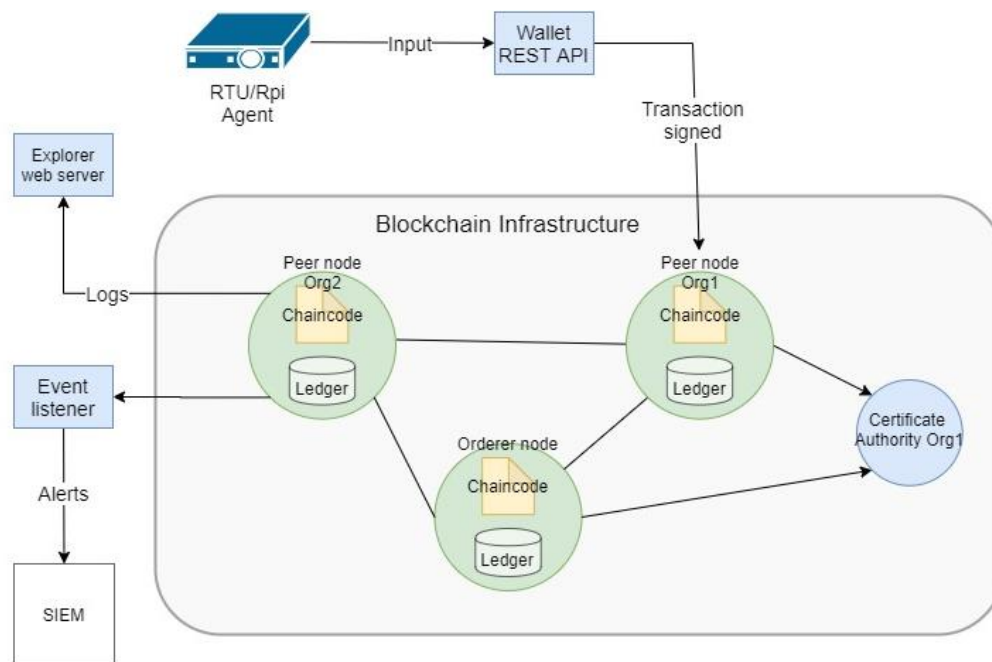


Figure 25. Blockchain Based Intrusion and Anomaly Detection system components

There are two types of requests in Hyperledger Fabric, invoke and query, the first one needs to change the ledger's state so it creates a new block, the second one is only reading function, so it does not mine a new block and it causes faster performance. Figure 22 shows the specification for each available function:

Table 22 Functions from the chaincode

| Name | Request type | Input (JSON) | | Description |
|-------------------------|--------------|-----------------------|----------------------------|--|
| | | Param | Type | |
| registerDevice | invoke | dvcID | string | Registers a new device to be monitored |
| getDeviceInfo | query | dvcID | string | Gets state from a previously registered device |
| getDeviceHistory | query | dvcID | string | Gets history from a device asset |
| connectivity | invoke | - | - | Checks last device update to be less than a period of time |
| registerHash | invoke | dvcID path hash | string string string | Registers a new hash related to an existing device |
| updateHash | invoke | hashID hash | string string | There are two input options |
| | | dvcID path hash | string string string | Updates state of a previously registered hash |

| | | | | |
|-----------------------|--------|--|-----------------------------|--|
| getHashInfo | query | hashID | string | Gets state from a previously registered hash |
| registerRecord | invoke | dvcID max (opt) min (opt) value (opt) | string int int int | Registers a new record related to an existing device |
| updateRecord | invoke | recordID value | string int | Updates state of a previously registered record |
| | | dvcID param value | string string int | |
| getRecordInfo | query | recordID | string | Gets state from a previously registered record |
| getAlertInfo | query | alertID | string | Gets state from an existing alert |

Tables 23-26 describe the structures that the chaincode manages:

Table 23 Device structure

| Device | | |
|----------------|-----------------|-------------------------------|
| Name | Type | Description |
| DvcID | string | Device unique identifier |
| Hashes | []*HashModel | Array of associated hashes |
| Records | []*RecordModel | Array of associated records |
| Alerts | []*AlertModel | Array of associated alerts |
| Status | bool | Indicates device availability |

Table 24 Hash structure

| Hash | | |
|---------------|--------|--------------------------------|
| Name | Type | Description |
| HashID | string | Hash unique identifier |
| DvcID | string | Associated device identifier |
| Path | string | Hashed file path in the device |
| Value | string | Last hash value received |

Table 25 Record structure

| Record | | |
|----------|---------|------------------------------|
| Name | Type | Description |
| RecordID | string | Record unique identifier |
| DvclD | string | Associated device identifier |
| Param | string | Monitored record description |
| Value | float32 | Last record value received |
| Max | float32 | Record maximum limitation |
| Min | float32 | Record minimum limitation |

Table 26 Alert structure

| Alert | | |
|-----------|---------|---|
| Name | Type | Description |
| AlertID | string | Alert unique identifier |
| DvclD | string | Associated device identifier |
| HashID | string | Associated hash identifier |
| RecordID | string | Associated record identifier |
| Class | string | Message describing the alert |
| PrevValue | string | Value read from ledger when anomaly was detected |
| RcvValue | string | Value received from client when anomaly was detected |
| Max | float32 | Reached maximum limitation when record anomaly was detected |
| Min | float32 | Reached minimum limitation when record anomaly was detected |

In order to clarify the behavior of the BIAD, Figure 26 presents a sequence diagram with the flow that follows BIAD in an operation.

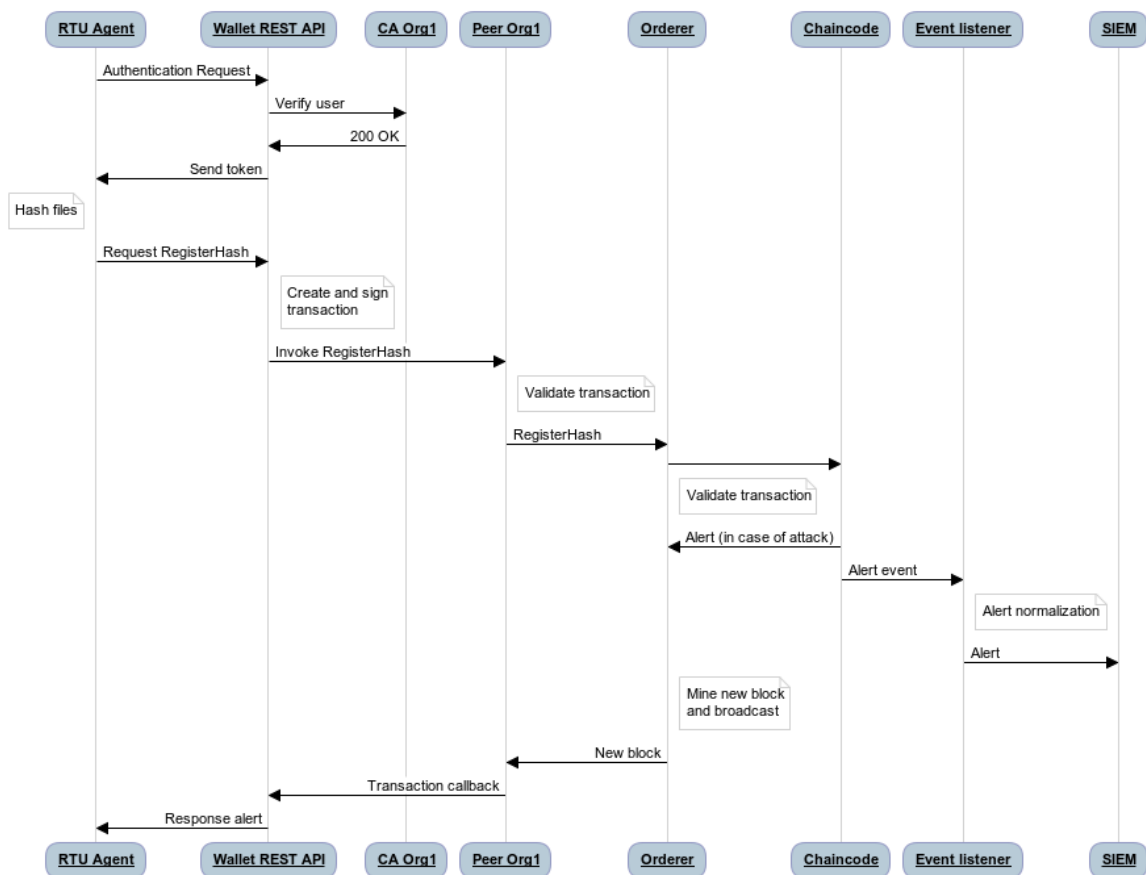


Figure 26: BIAD behaviour model

4.2 Interfaces model

4.2.1 External interfaces with other SDN-microSENSE application plane modules

4.2.1.1 Communication of e-auction module with EMO

The OTSC EMO tool provides a Rest API to validate if changes proposed by the e-auction module of the Energy Trading System are feasible from an electrical point of view:

Table 27: EMO – e-auction interface

| Interface name: OTSC EMO – e-auction | |
|--------------------------------------|---|
| Description | This interface lets us to evaluate the feasibility in electrical terms from a set of proposed changes for the grid model. |
| Component providing the interface | OTSC EMO |
| Consumer components | e-auction |
| Used Technology | REST API |
| State | Developed |
| Input data | List of energy trading actions to validate Example: |

| | |
|--------------------|--|
| | <pre>[{ "id": 1, "energy": 40.0, "time_action": "18/08/2020 09:00:00.000" "time": 15.0 "seller_node": "prosumer1" "buyer_node": "prosumer4" "seller_baseline_power": 10.0, "buyer_baseline_power": 10.0 }, { "id": 2, "energy": 20.0, "time_action": "18/08/2020 09:00:00.000" "time": 15.0 "seller_node": "prosumer2" "buyer_node": "prosumer3" "seller_baseline_power": 5.0, "buyer_baseline_power": 15.0 }]</pre> |
| Output data | Feasibility or not of the proposed changes regarding the electrical point of view. Example: <pre>[{ "id": 1, "result": "OK" }, { "id": 2, "result": "KO" }]</pre> |
| API URL | http://172.21.0.22:8000/validate (internal URL). As it is going to be accessed from out of the OTSC EMO virtual machine (VM15), the important thing here is the URL for external access: http://IP_virtual_machine_for_OTSC_EMO:8888/validate |
| Constraints | None |
| Responsibilities | IREC |
| Documentation link | To Be Determined |

In Figure 27, there is an example about how to do a call to this Rest API:

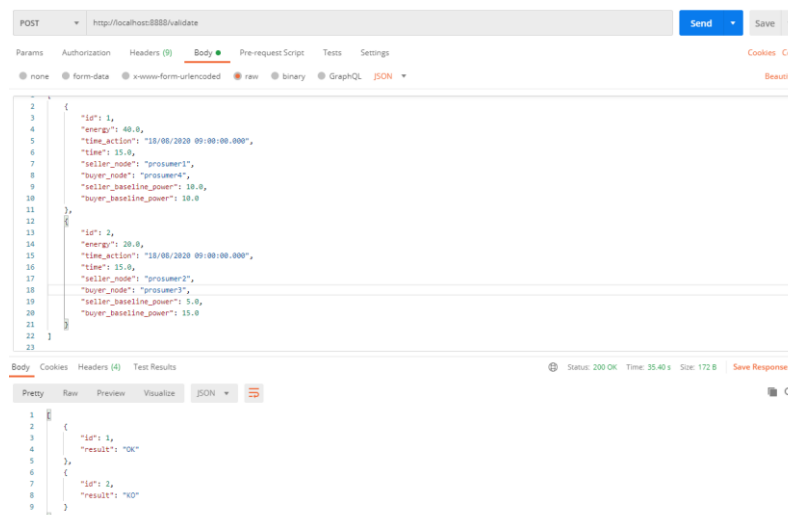


Figure 27 Example request of Market Validator

4.2.1.2 Communication of e-auction module with S-RAF

The e-auction module interacts with S-RAF for acquiring the latest risk assessment results, as those have been generated considering the vulnerabilities of the infrastructure and the threats detected by the XL-EPDS infrastructure. As presented in Figure 1 for the blockchain-based energy trading system within the SDN-microSENSE architecture, the e-auction module considers the information of the risk levels, which is generated on asset-basis, in order to decide whether to deny participation to the market to organizations that face cyber risks.

To do so, the e-auction module subscribes to the KAFKA component of S-RAF for consuming the risk assessment information. Table 28 summarizes the necessary interface details for materializing this interaction.

Table 28: e-auction – SRAF interface

| Interface name: e-auction module – S-RAF | |
|--|--|
| Description | The objective of this interface is to make available the risk assessment result to the e-auction module. |
| Component providing the interface | S-RAF |
| Consumer components | e-auction module |
| Used Technology | KAFKA |
| State | Developed |
| Input data | Events from XL-EPDS infrastructure (via CIS). |
| Output data | Risk assessment results generated on asset-basis containing information about the risk level of assets. |
| API URL | N/A |
| Constraints | None |
| Responsibilities | UBITECH to provide more information to the developing team of e-auction module to ease integration. |
| Documentation link | More details can be found in D3.5. |

4.2.1.3 Communication of Blockchain-based Intrusion and Anomaly Detection with XL-SIEM

XL-SIEM uses syslog as a mechanism for receiving information. The provided script *rsyslog.sh* prepares the rsyslog to send information to the XL-SIEM. Once the rsyslog has been properly installed and configured, it is necessary to install the module which listens for alerts from the BIAD and converts them to a readable format for the XL-SIEM. To run this module, it is required to have NodeJS installed in the SO as well as some libraries with *npm install*.

This component listens for the alerts generated in the Blockchain based Intrusion and Anomaly Detection module, that have a predetermined format and converts them into a format, which is understood by the XL-SIEM. The format of the alert is not specified in this deliverable, in order to ensure the protection of the sensitive information.

The word “BIAD” in the alert will be automatically included by rsyslog to inform XL-SIEM the detector of the alert.

Focusing on the technology, the connection with the XL-SIEM is made using NodeJS as programming language as it has been introduced before. The script listens for events from the Hyperledger Fabric Network and writes a new line in `/var/log/biad.log` file. Then, the rsyslog daemon detects the changes in this file and sends the new line to the XL-SIEM. This software can be easily adapted to send information to any endpoint, not necessarily the XL-SIEM.

Table 29: BIAD - XL-SIEM interface

| Interface name: Blockchain based Intrusion detection - XL-SIEM | |
|--|--|
| Description | The objective of this interface is to send the alerts generated by the Blockchain based Intrusion detection module to XL-SIEM. |
| Component providing the interface | XL-SIEM |
| Consumer components | Blockchain Based Intrusion and Anomaly Detection (BIAD) |
| Used Technology | Rsyslog |
| State | Developed |
| Input data | Events from the Hyperledger Fabric Network |
| Output data | Alert in the format that expected by the XL-SIEM |
| API URL | NA |
| Constraints | None |
| Responsibilities | TECNALIA to develop the module to communicate to the XL-SIEM |
| Documentation link | NA |

4.2.2 External interfaces with SDN-microSENSE infrastructure plane modules

4.2.2.1 Connection of e-auction module with RPI of smart meter

Each prosumer participating in the network is represented in the e-auction processes by an agent, that utilizes information such as the security status of the prosumer’s smart meter and the energy balance forecasting provided by external tools, in order to take decisions regarding the role in the market (auctioneer/bidder), as well as the price of selling/buying. The flow chart in Figure 28, summarizes the algorithms executed by the agents in each time slot:

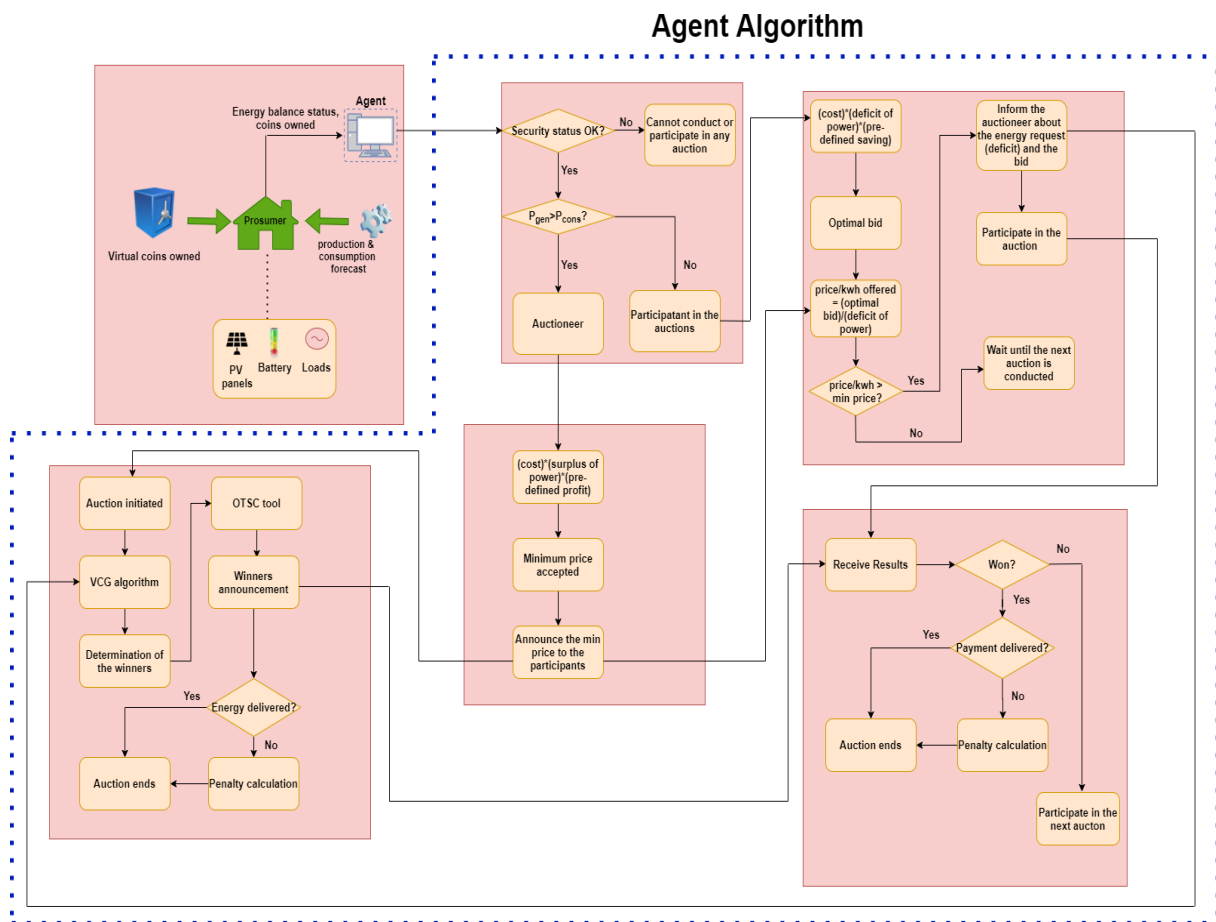


Figure 28: Agent algorithm

It must be noted that in Figure 24 the penalty mechanism functionality is included, which is not part of the current version of the Energy Trading Framework. Apart from the penalty mechanism, all the procedures demonstrated in the flow chart are implemented in the current version. The flow chart was designed with the aim to provide the reader with the general overview of the algorithms run by the agents.

The flow chart in Figure 29, demonstrates the algorithm followed by the agents in order to determine the way in which they will pay the penalties imposed to them, and refers only to the extension of the Energy Trading Framework, in which the penalty mechanism will be included.

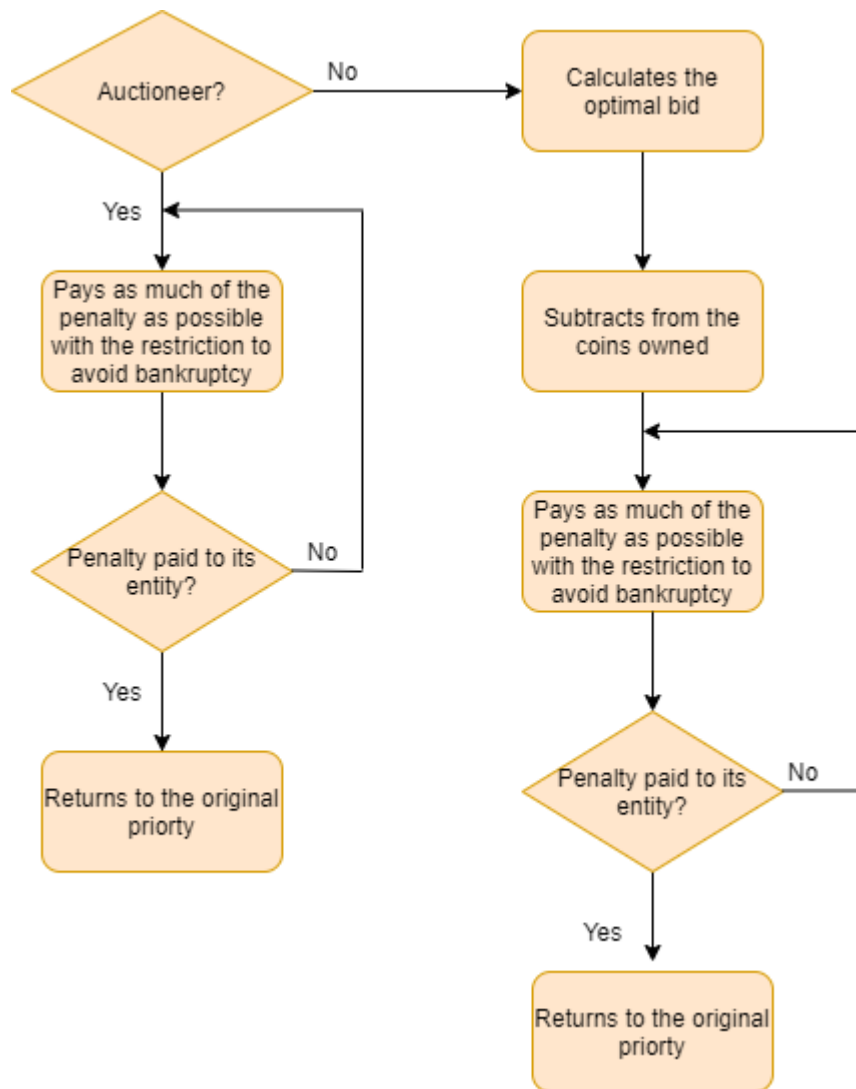


Figure 29: Penalty paying algorithm

4.2.2.2 Connection of Blockchain Based Intrusion and Anomaly Detection module with RPI of smart meter

The Blockchain-Based Intrusion and Anomaly Detection (BIAD) module requires an agent to send information to the Blockchain part (See Section 4.1.2). This agent can be installed in any device (in the context of the SDN-microSENSE project Raspberry PI based smart meter is going to be used in order to be able to do the proof of concept) that is about to be monitored and it must be configured to connect with the BIAD. Further information about the installation process is provided in section 6.2.

The BIAD agent has been developed using C programming language, so it can be compiled in any device having a C compiler. Furthermore, C is one of the fastest languages, which is a crucial aspect, considering that the target devices are usually constraint in terms of processor power, RAM memory, or even battery. Consequently, developing a lightweight agent is a fundamental requirement within the BIAD component.

As part of the configuration, some file paths must be given to the agent. The BIAD agent then systematically looks for these files in the filesystem and hashes them. Once this hash has been calculated, the next step is to submit it to the Blockchain and perform the comparison as it has been previously discussed. The BIAD agent can bind as many files as required and with any format, so different files can be monitored, such as: configuration files, log files or even the whole firmware.

Furthermore, the agent checks useful kernel information, such as: number of processes, RAM usage, and uptime. This information is also sent to the BIAD – peer node to be analyzed. It is important to notice that the BIAD is agnostic when it comes to this information, because is the BIAD who adds the intelligence to this information. The agent works only as a lightweight daemon who passes information to the BIAD.

As it will be explained in the section 6.2, all the configuration process is made directly over the code before compilation, so no information is retrieved from other sources. By doing it this way, we can later remove every evidence of the agent installation, except the executable, so it can be more easily hidden in the system. A process-hiding mechanism has also been included in the agent, to hide the subsequent process from being shown with the rest of system processes and this system has been tested in Debian systems. This strategy is considered security-by-obscurity, but we have considered that any extra occultation layer is welcomed. At the end, time is crucial in the solution, and the most time an attacker spends in detecting the agent, the better.

5. System verification

To verify that the Blockchain-based Energy Trading System meets its requirements, several unit tests have been defined and executed. Unit testing is an integral part of TDD (test-driven development) procedures, as it facilitates designing robust software components by finding and fixing defects in the early stages of software development lifecycle. The flow-chart of Figure 26, presents the procedure of TDD.

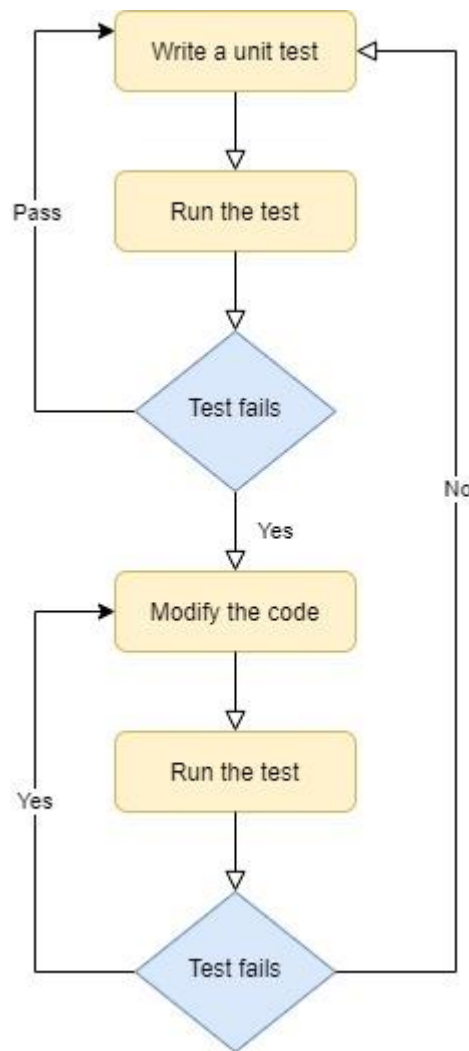


Figure 26: TDD procedure

In sections 5.1 and 5.2, the unit tests for both e-auction module and Blockchain Based Intrusion and Anomaly Detection module are presented, using test case tables that include description of the unit test, the related requirements for which the unit test was written, the priority of the test for the development of the software, the pre-conditions that should be fulfilled to run the test, the test steps that compose the unit test, the input data for each step, the results after executing each step and the result of the whole unit test.

5.1 E-auction module unit tests

Table 30: E-AUCTION_01 unit test

| Test Case ID | E-AUCTION_01 | Component | E-auction module of energy trading framework |
|--------------|---|-----------|--|
| Description | ESCO company is responsible for managing ERC20 token accounts. This test showcases the ability of ESCO organization to initialize a token, create and update user accounts. | | |

| | | | |
|------------------|--|-----------|-------|
| Req ID | FR-UC4-04 | Priority | High |
| Prepared by | CERTH | Tested by | CERTH |
| Pre-condition(s) | <ul style="list-style-type: none">Each component of the e-auction has to be up and listening on their respective port, which includes at least an orderer, a peer for the ESCO organization, one or more peers for prosumers and a couchdb for each peer.The ERC20 chaincode must be installed and instantiated on the e-auction channel of the blockchain network.The application of each organization is running | | |
| Test steps | | | |
| 1 | The peer of ESCO organization sends a POST HTTP request to initialize a new token, with the name of the token, the symbol of the token and the amount of token units that correspond to the total supply. | | |
| 2 | The peers of each prosumer organization (Org1, Org2 and Org3), apply for token account with an HTTP POST request, including the name and the symbol of the token in the request body. | | |
| 3 | After applying each prosumer should be able to retrieve the balance of its own with an HTTP GET request with the name and the symbol of the token as URL parameters. The initial balance should equal to 0. | | |
| 4 | The ESCO retrieves the list of applicant organizations, through an HTTP GET request with the name and symbol of the token as URL parameters | | |
| 5 | The ESCO organization's peer updates the balances of the applicants with 500 tokens for each account as an initial amount. This action is performed with an HTTP POST request with the name and symbol of the token, the amount of the tokens and the prosumer's identification information (MSP ID and CertID). | | |
| 5 | At this stage, each prosumer retrieves its balance, that should be equal to 500 tokens. | | |
| Input data | <ul style="list-style-type: none">Step1 <pre>{ "Name": "tok", "Symbol": "BT", "TotalSupply": 1000000 }</pre>Step2 <pre>{</pre> | | |

```
"Name": "tok",
```

```
"Symbol": "BT"
```

```
}
```

- **Step3**

```
name=tok
```

```
symbol=BT
```

```
msplD=Org1MSP
```

```
certID=eDU...VVT
```

- **Step4**

```
name=tok
```

```
symbol=BT
```

- **Step5**

```
{
```

```
  "Name": "tok",
```

```
  "Symbol": "BT",
```

```
  "Amount": 500,
```

```
  "From": {
```

```
    "MspId": "Org1MSP",
```

```
    "CertId": "eDU...VVT"
```

```
  }
```

```
}
```

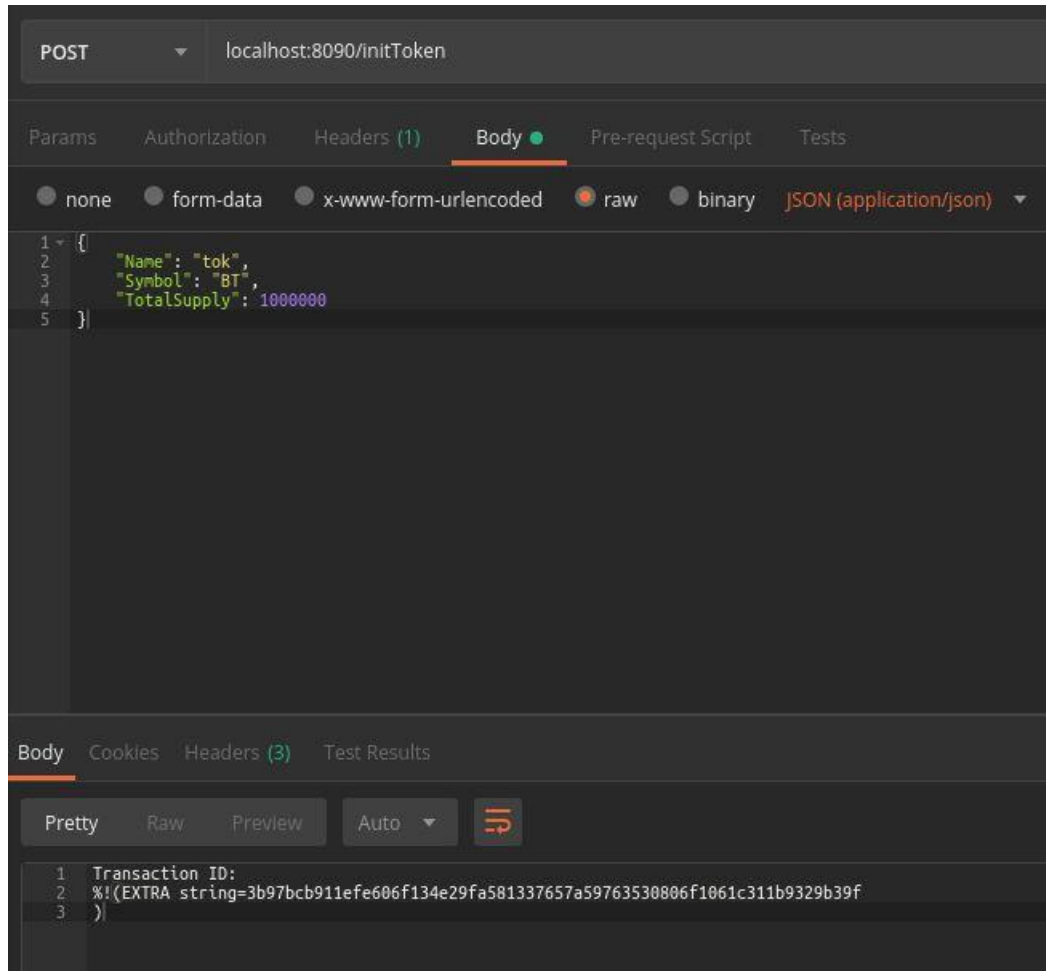
- **Step6**

```
name=tok
```

```
symbol=BT
```

Result

- Step1



POST localhost:8090/initToken

Params Authorization Headers (1) **Body** Pre-request Script Tests

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary JSON (application/json)

```

1 {
2   "Name": "tok",
3   "Symbol": "BT",
4   "TotalSupply": 1000000
5 }

```

Body Cookies Headers (3) Test Results

Pretty Raw Preview Auto

```

1 Transaction ID:
2 %!(EXTRA string=3b97bcb911efe606f134e29fa581337657a59763530806f1061c311b9329b39f
3 )

```

- Step2

localhost:8091/apply

POST localhost:8091/apply

Params Authorization Headers (1) **Body** Pre-request Script Tests

none form-data x-www-form-urlencoded **raw** binary JSON (application/json)

```

1 {
2   "Name": "tok",
3   "Symbol": "BT"
4 }

```

Body Cookies Headers (3) Test Results

Pretty Raw Preview Auto

```

1 Transaction ID:
2 %!(EXTRA string=a6bdd5c53a5591290259a2d284a244e8748c76542e62812665bb2db4f1daec8a
3 )

```

- Step3

GET localhost:8091/getBalance?name=tok&symbol=BT&mspiID=Org1MSP&certID=eDUwOT06Q049VXNlcjFAb3JnMS5leGFtcGxLmNvbSxMPVNhbiBGcmFuY2

Params Authorization Headers Body Pre-request Script Tests

| KEY | VALUE |
|--|---|
| <input checked="" type="checkbox"/> name | tok |
| <input checked="" type="checkbox"/> symbol | BT |
| <input checked="" type="checkbox"/> mspiID | Org1MSP |
| <input checked="" type="checkbox"/> certID | eDUwOT06Q049VXNlcjFAb3JnMS5leGFtcGxLmNvbSxMPVNhbiBGcmFuY2 |
| Key | Value |

Body Cookies Headers (3) Test Results

Pretty Raw Preview JSON

```

1 {
2   "Balance": "0"
3 }

```

- Step4

[illegible]

- **Step5**

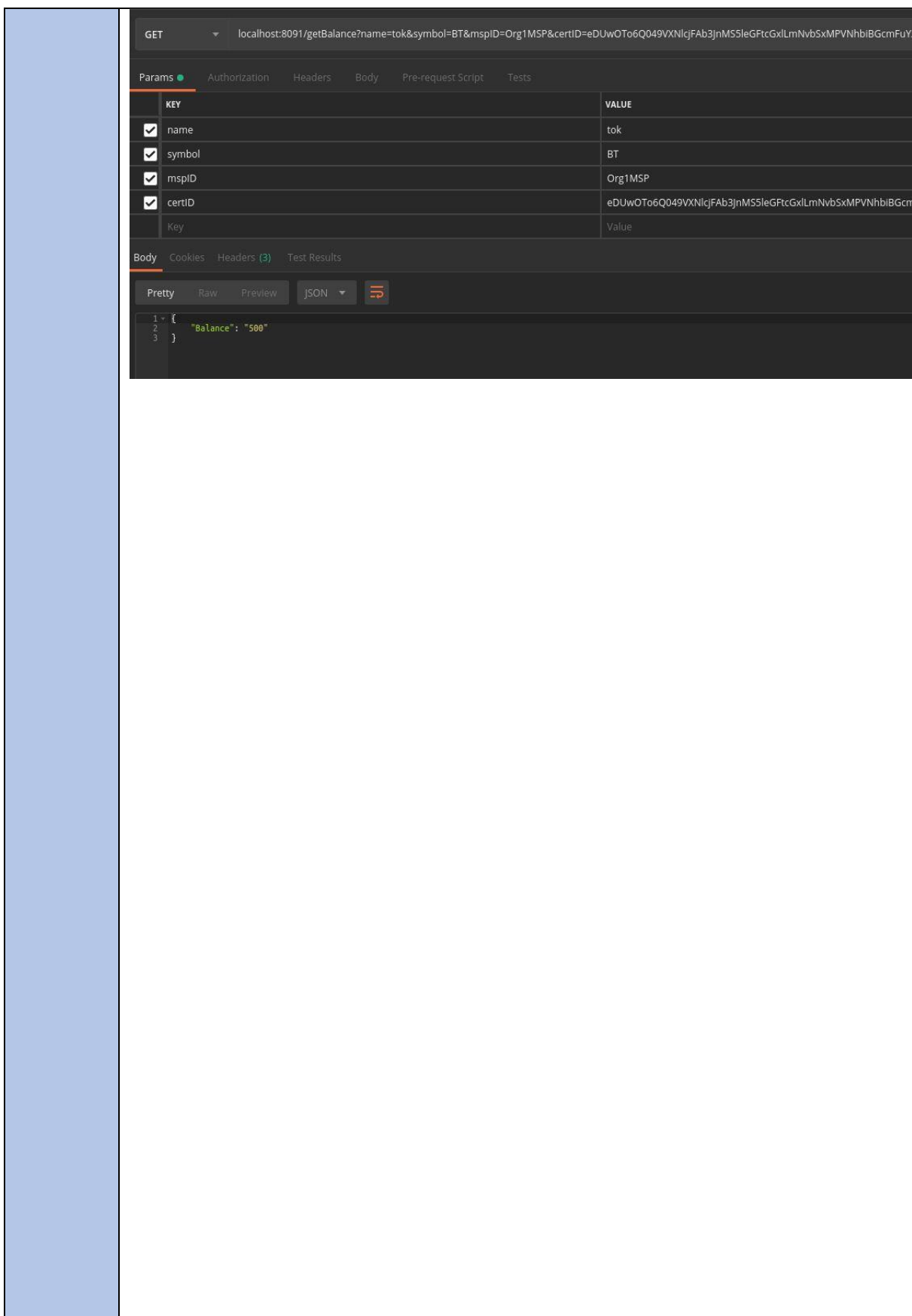
The screenshot displays a REST client interface with a POST request to `localhost:8090/updateBalance`. The **Body** tab is selected, showing a JSON payload:

```
1 {
2   "Name": "tok",
3   "Symbol": "BT",
4   "Amount": 500,
5   "From": {
6     "MspId": "Org1MSP",
7     "CertId": "e0Uw0To6Q0849VXNlcjFab3JnMSS1eGFtcGxLLmNvbSxMPVNhbiBGMFuY2ZyY28sU1Q9Q2FsaWZvcn5pYSxDP"
8   }
9 }
```

Below the request, the **Body** tab is selected in the response section, showing the response body:

```
1 Transaction ID:
2 %!(EXTRA string=a397a02ba14306dc5a84378f82e05d2cfac53ac8a9210d764963d8ad7041e1b6
3 )
```

- **Step6**




GET localhost:8091/getBalance?name=tok&symbol=BT&mspiD=Org1MSP&certID=eDUwOT6Q049VXNlcjFab3JnMS5leGFtcGxLLmNvbSxMPVNhbiBGcmFuY2

Params Authorization Headers Body Pre-request Script Tests

| KEY | VALUE |
|--|---|
| <input checked="" type="checkbox"/> name | tok |
| <input checked="" type="checkbox"/> symbol | BT |
| <input checked="" type="checkbox"/> mspiD | Org1MSP |
| <input checked="" type="checkbox"/> certID | eDUwOT6Q049VXNlcjFab3JnMS5leGFtcGxLLmNvbSxMPVNhbiBGcm |
| Key | Value |

Body Cookies Headers (3) Test Results

Pretty Raw Preview JSON 

```

1 {
2   "Balance": "500"
3 }

```

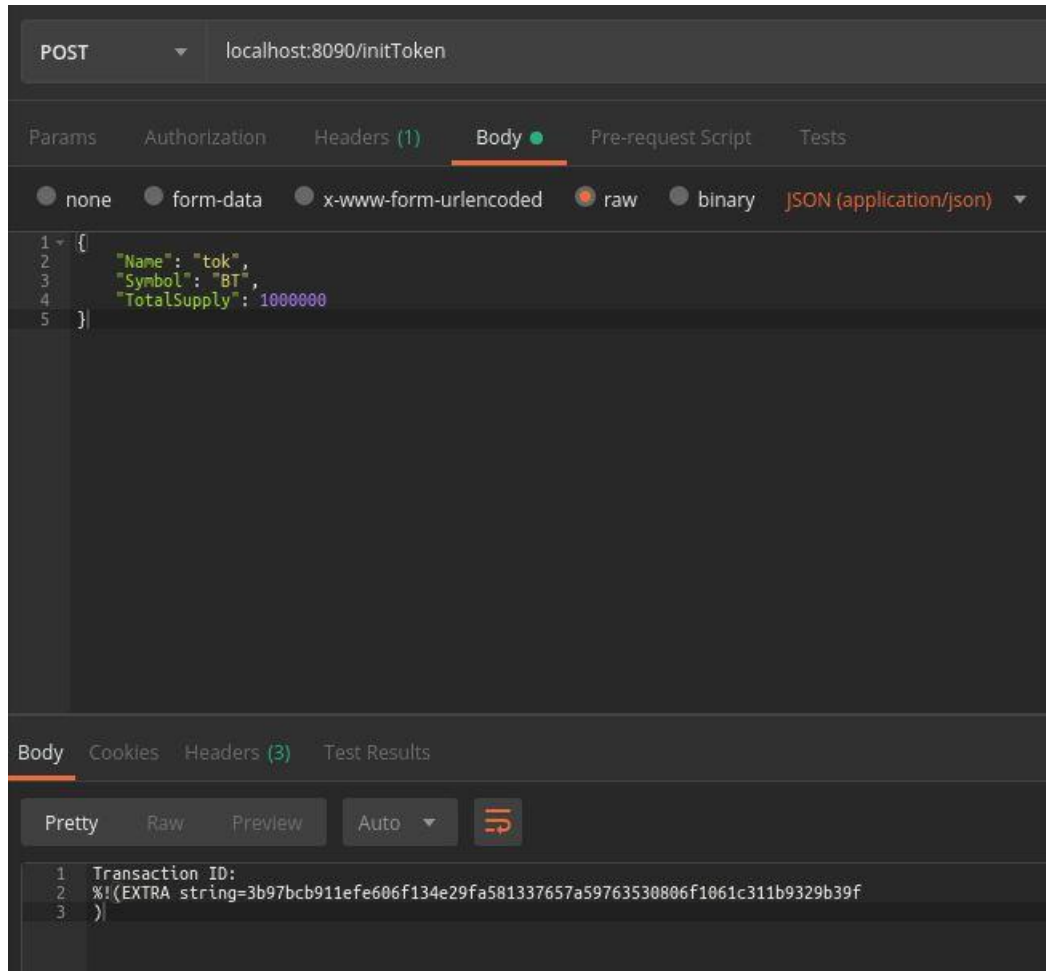
| | |
|------------------|----------|
| Test Case Result | Achieved |
|------------------|----------|

| | | | |
|------------------|--|-----------|--|
| Test Case ID | E-AUCTION_01 | Component | E-auction module of energy trading framework |
| Description | ESCO company is responsible for managing ERC20 token accounts. This test showcases the ability of ESCO organization to initialize a token, create and update user accounts. | | |
| Req ID | FR-UC4-04 | Priority | High |
| Prepared by | CERTH | Tested by | CERTH |
| Pre-condition(s) | <ul style="list-style-type: none">Each component of the e-auction has to be up and listening on their respective port, which includes at least an orderer, a peer for the ESCO organization, one or more peers for prosumers and a couchdb for each peer.The ERC20 chaincode must be installed and instantiated on the e-auction channel of the blockchain network.The application of each organization is running | | |
| Test steps | | | |
| 1 | The peer of ESCO organization sends a POST HTTP request to initialize a new token, with the name of the token, the symbol of the token and the amount of token units that correspond to the total supply. | | |
| 2 | The peers of each prosumer organization (Org1, Org2 and Org3), apply for token account with an HTTP POST request, including the name and the symbol of the token in the request body. | | |
| 3 | After applying each prosumer should be able to retrieve the balance of its own with an HTTP GET request with the name and the symbol of the token as URL parameters. The initial balance should equal to 0. | | |
| 4 | The ESCO retrieves the list of applicant organizations, through an HTTP GET request with the name and symbol of the token as URL parameters | | |
| 5 | The ESCO organization's peer updates the balances of the applicants with 500 tokens for each account as an initial amount. This action is performed with an HTTP POST request with the name and symbol of the token, the amount of the tokens and the prosumer's identification information (MSP ID and CertID). | | |
| 5 | At this stage, each prosumer retrieves its balance, that should be equal to 500 tokens. | | |

| | |
|------------|--|
| Input data | <ul style="list-style-type: none"> • Step1 <pre>{ "Name": "tok", "Symbol": "BT", "TotalSupply": 1000000 }</pre> • Step2 <pre>{ "Name": "tok", "Symbol": "BT" }</pre> • Step3 <pre>name=tok symbol=BT mspID=Org1MSP certID=eDU...VVT</pre> • Step4 <pre>name=tok symbol=BT</pre> • Step5 <pre>{ "Name": "tok", "Symbol": "BT", "Amount": 500, "From": { "MspId": "Org1MSP", "CertId": "eDU...VVT" } }</pre> • Step6 <pre>name=tok symbol=BT</pre> |
|------------|--|

Result

- Step1



POST localhost:8090/initToken

Params Authorization Headers (1) **Body** Pre-request Script Tests

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary JSON (application/json)

```

1 {
2   "Name": "tok",
3   "Symbol": "BT",
4   "TotalSupply": 1000000
5 }
```

Body Cookies Headers (3) Test Results

Pretty Raw Preview Auto

```

1 Transaction ID:
2 %!(EXTRA string=3b97bcb911efe606f134e29fa581337657a59763530806f1061c311b9329b39f
3 )
```

- Step2

localhost:8091/apply

POST localhost:8091/apply

Params Authorization Headers (1) **Body** Pre-request Script Tests

none form-data x-www-form-urlencoded **raw** binary JSON (application/json)

```

1 {
2   "Name": "tok",
3   "Symbol": "BT"
4 }

```

Body Cookies Headers (3) Test Results

Pretty Raw Preview Auto

```

1 Transaction ID:
2 %!(EXTRA string=a6bdd5c53a5591290259a2d284a244e8748c76542e62812665bb2db4f1daec8a
3 )

```

- Step3

GET localhost:8091/getBalance?name=tok&symbol=BT&mspiID=Org1MSP&certID=eDUwOT06Q049VXNlcjFAb3JnMS5leGFtcGxLmNvbSxMPVNhbiBGcmFuY2

Params Authorization Headers Body Pre-request Script Tests

| KEY | VALUE |
|--|---|
| <input checked="" type="checkbox"/> name | tok |
| <input checked="" type="checkbox"/> symbol | BT |
| <input checked="" type="checkbox"/> mspiID | Org1MSP |
| <input checked="" type="checkbox"/> certID | eDUwOT06Q049VXNlcjFAb3JnMS5leGFtcGxLmNvbSxMPVNhbiBGcmFuY2 |
| Key | Value |

Body Cookies Headers (3) Test Results

Pretty Raw Preview JSON

```

1 {
2   "Balance": "0"
3 }

```

- Step4

[illegible]

- **Step5**

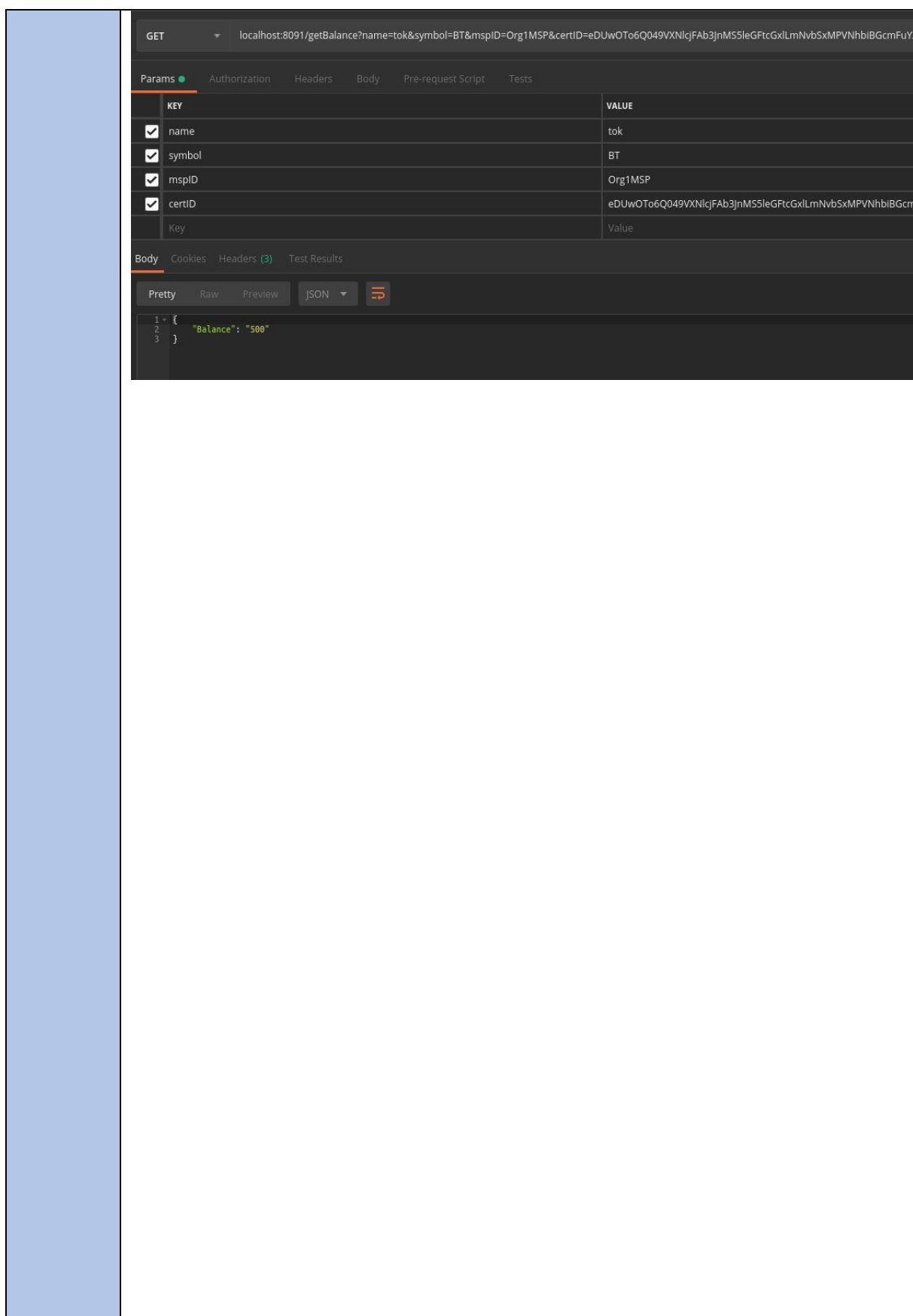
The screenshot displays a REST client interface with a POST request to `localhost:8090/updateBalance`. The **Body** tab is selected, showing a JSON payload:

```
1 {  
2   "Name": "tok",  
3   "Symbol": "BT",  
4   "Amount": 500,  
5   "From": {  
6     "MspId": "Org1MSP",  
7     "CertId": "eDUwOT06Q049VXNlcjFAB3JnMSS1eGFtcGxLLmNvbSxMPVNhbi8GcmFuY2ZlY28sU1Q9Q2FsaWZvcn5pYSxDP"  
8   }  
9 }
```

Below the request, the **Body** tab is selected in the response section, showing the response body in a "Pretty" format:

```
1 Transaction ID:  
2 %!(EXTRA string=a397a02ba14306dc5a84378f82e05d2cfac53ac8a9210d764963d8ad7041e1b6  
3 )
```

- **Step6**




GET localhost:8091/getBalance?name=tok&symbol=BT&mspiD=Org1MSP&certID=eDUwOT6Q049VXNlcjFab3JnMS5leGFtcGxILmNvbSxMPVNhbiBGcmFuY2

Params Authorization Headers Body Pre-request Script Tests

| KEY | VALUE |
|--|---|
| <input checked="" type="checkbox"/> name | tok |
| <input checked="" type="checkbox"/> symbol | BT |
| <input checked="" type="checkbox"/> mspiD | Org1MSP |
| <input checked="" type="checkbox"/> certID | eDUwOT6Q049VXNlcjFab3JnMS5leGFtcGxILmNvbSxMPVNhbiBGcmFuY2 |
| Key | Value |

Body Cookies Headers (3) Test Results

Pretty Raw Preview JSON 

```

1 {
2   "Balance": "500"
3 }

```

| | |
|------------------|----------|
| Test Case Result | Achieved |
|------------------|----------|

Table 31: E-AUCTION_02 unit test

| | | | |
|-------------------|---|-----------|--|
| Test Case ID | E-AUCTION_02 | Component | E-auction module of energy trading framework |
| Description | A priority table is managed by the related chaincode that keeps the total contribution to the markets for each organization. This test, showcases the initialization of the priority table by the ESCO organization peer and how organizations are registered to the priority table after applying to a market. | | |
| Req ID | FR-UC4-02 | Priority | High |
| Prepared by | CERTH | Tested by | CERTH |
| Pre-condition (s) | <ul style="list-style-type: none">Each component of the e-auction has to be up and listening on their respective port, which includes at least an orderer, a peer for the ESCO organization, one or more peers for prosumers and a couchdb for each peer.The Priority Table and the Market chaincodes must be installed and instantiated on the e-auction channel of the blockchain network.The application of each organization is runningA market has been initialized | | |
| Test steps | | | |
| 1 | The ESCO organization’s peer initiates a new priority table using an HTTP POST request. | | |
| 2 | ESCO peer retrieves the priority table with an HTTP GET request. The initial participants and market contribution lists should be empty. | | |
| 3 | Peers of prosumer organizations (Org1 and Org2) apply to a market with an HTTP POST request including the ID of a market that they are applying for, the ID of the priority table and their role in the market (Org1 as Auctioneer and Org2 as Bidder) | | |
| 4 | ESCO peer retrieves the priority table with a GET request. The organizations that have applied to a market should have been added to the lists of participants and contributions of the priority table and their initial contribution should equal to 0. | | |
| Input data | <ul style="list-style-type: none">Step1 HTTP POST request with empty request bodyStep2 HTTP GET request without parameters | | |

- **Step3**

```
{  
  "ID": "8680ccf55f5c4d03845ee82d4b1c35e5",  
  "PriorityTable": "6687732881c54b60a0168aefd4c6cf60",  
  "Role": "Auctioneer"  
}
```

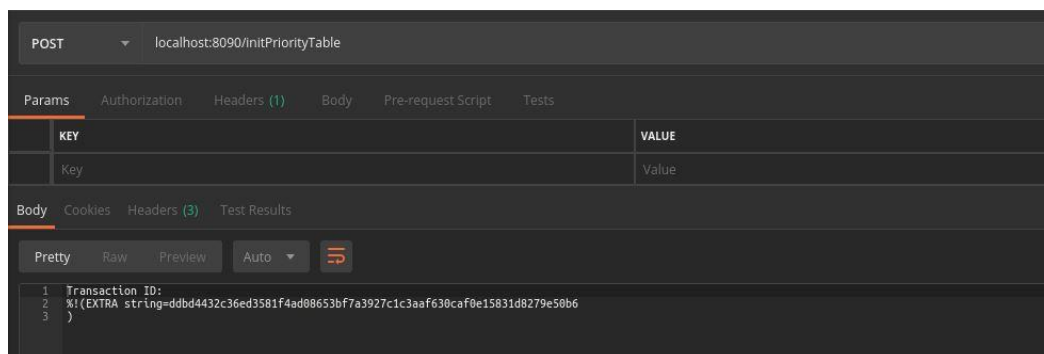
```
{  
  "ID": "8680ccf55f5c4d03845ee82d4b1c35e5",  
  "PriorityTable": "6687732881c54b60a0168aefd4c6cf60",  
  "Role": "Bidder"  
}
```

- **Step4**

HTTP GET request without parameters

Result

• Step 1



POST localhost:8090/initPriorityTable

Params Authorization Headers (1) Body Pre-request Script Tests

| KEY | VALUE |
|-----|-------|
| Key | Value |

Body Cookies Headers (3) Test Results

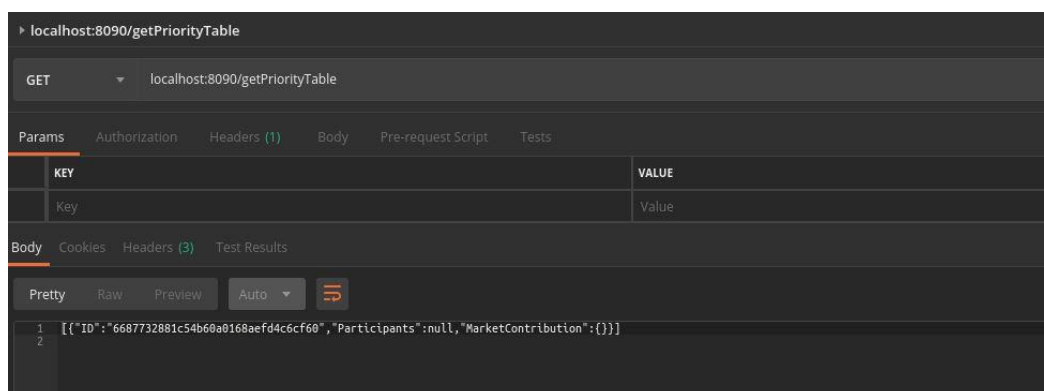
Pretty Raw Preview Auto

```

1 Transaction ID:
2 %!(EXTRA string=ddb4432c36ed3581f4ad08653bf7a3927c1c3aaf630caf0e15831d8279e50b6
3 )

```

• Step 2



localhost:8090/getPriorityTable

GET localhost:8090/getPriorityTable

Params Authorization Headers (1) Body Pre-request Script Tests

| KEY | VALUE |
|-----|-------|
| Key | Value |

Body Cookies Headers (3) Test Results

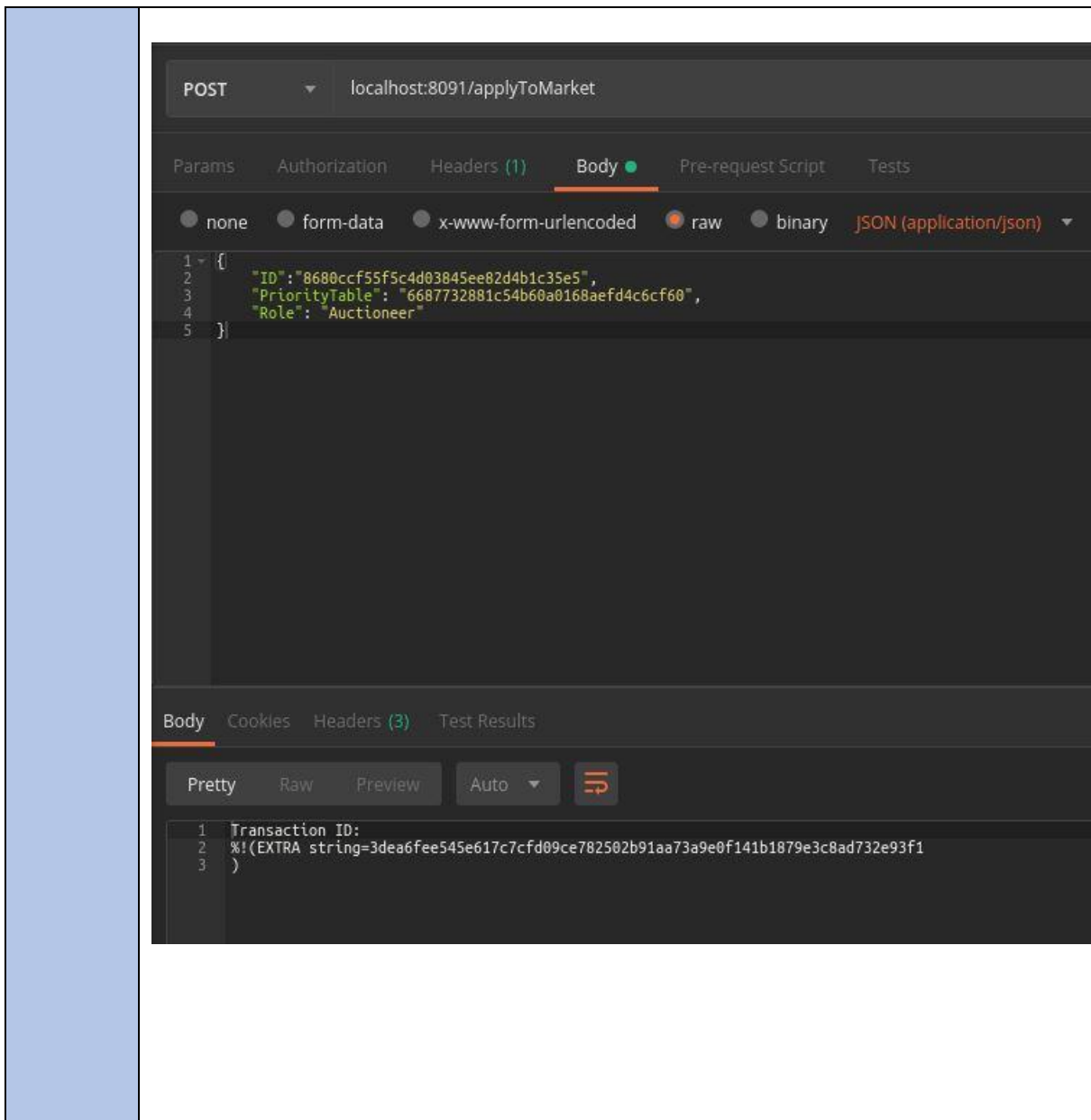
Pretty Raw Preview Auto

```

1 [{"ID": "6687732881c54b60a0168aefd4c6cf60", "Participants": null, "MarketContribution": {}}]
2
3

```

• Step 3

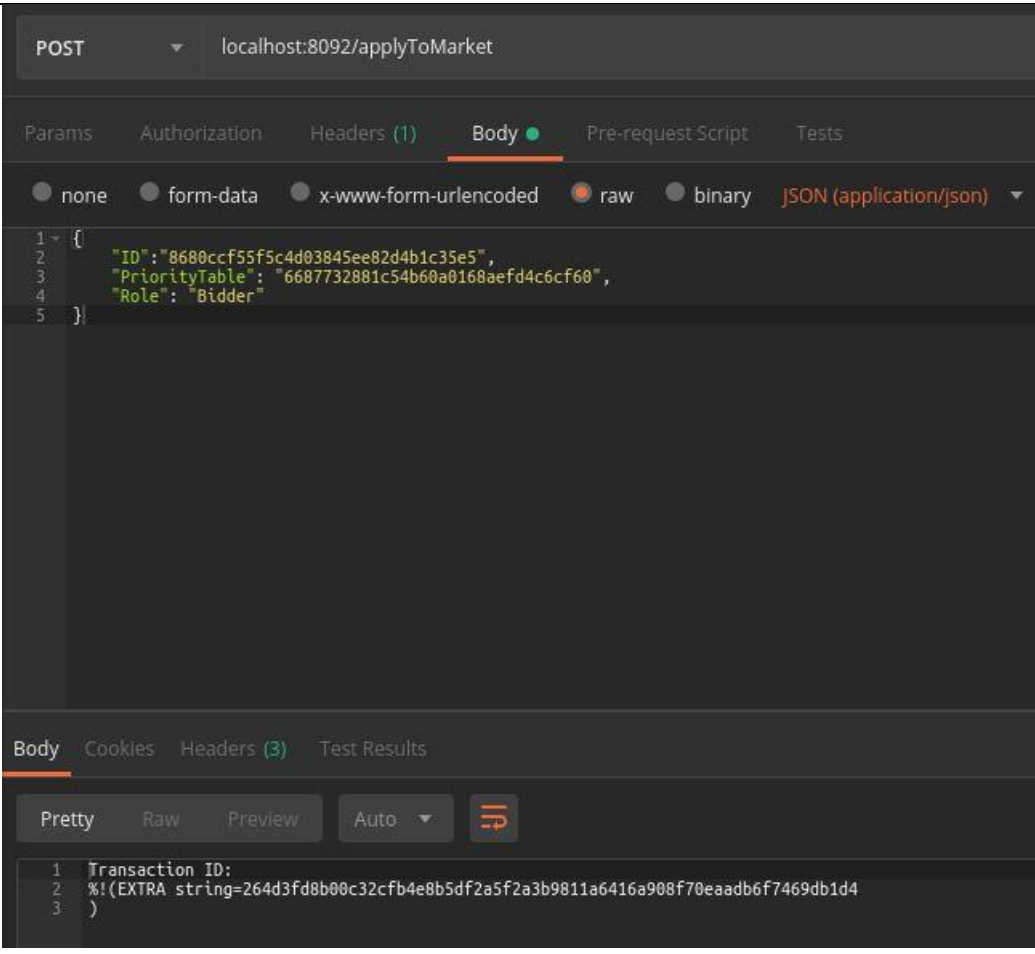
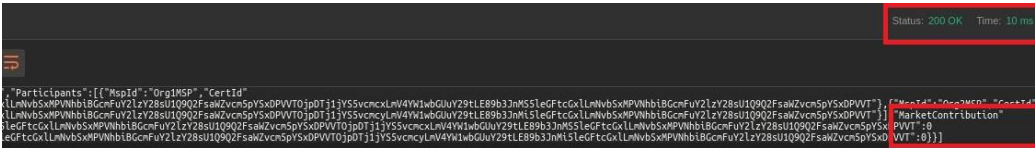


The screenshot displays a REST client interface with the following details:

- Method:** POST
- URL:** localhost:8091/applyToMarket
- Body Type:** raw (selected), with a content type of `JSON (application/json)`.
- Request Body (JSON):**

```
{
  "ID": "8680ccf55f5c4d03845ee82d4b1c35e5",
  "PriorityTable": "6687732881c54b60a0168aefd4c6cf60",
  "Role": "Auctioneer"
}
```
- Response Body (Pretty):**

```
1 Transaction ID:
2 %!(EXTRA string=3dea6fee545e617c7cfd09ce782502b91aa73a9e0f141b1879e3c8ad732e93f1
3 )
```

| | |
|------------------|--|
| |  <p>• Step4</p>  |
| Test Case Result | Achieved |

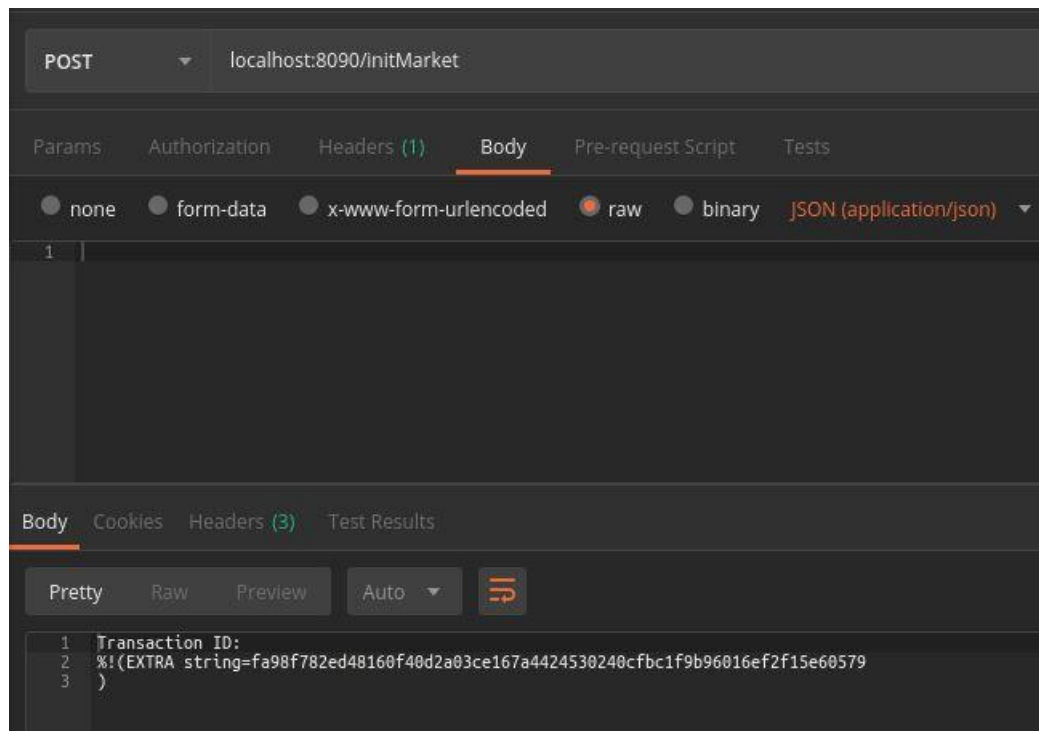
| | | | |
|--------------|--|-----------|--|
| Test Case ID | E-AUCTION_03 | Component | E-auction module of energy trading framework |
| Description | Every 15 minutes, a new market is initialized. The organizations have a specific time slot (1 minute) to apply for that market and declare their roles. After the deadline passes then only the registered organizations will take part in the market. This test | | |

| | | | |
|------------------|--|-----------|-------|
| | showcases the initialization of a market by the ESCO organization and successful/unsuccessful applications from prosumers' peers. | | |
| Req ID | FR-UC4-02 | Priority | High |
| Prepared by | CERTH | Tested by | CERTH |
| Pre-condition(s) | <ul style="list-style-type: none">Each component of the e-auction has to be up and listening on their respective port, which includes at least an orderer, a peer for the ESCO organization, one or more peers for prosumers and a couchdb for each peer.The Market chaincode must be installed and instantiated on the e-auction channel of the blockchain network.The application of each organization is runningThe priority table has been initialized and the ID is known to the organizations | | |
| Test steps | | | |
| 1 | The ESCO organization's peer initiates a new market, with an HTTP POST request with empty request body. | | |
| 2 | Peer of prosumer Org1 applies to the market with an HTTP POST request including the ID of a market, the ID of the priority table and 'Auctioneer' role, successfully. | | |
| 3 | Peer of prosumer Org2 applies to the market with an HTTP POST request including the ID of a market, the ID of the priority table and 'Bidder' role, successfully. | | |
| 4 | Peer of prosumer Org3 applies to the market with an HTTP POST request including the ID of a market, the ID of the priority table and 'Bidder' role, unsuccessfully because the application deadline has already passed. | | |
| 5 | The ESCO organization's peer retrieves the markets list with an HTTP GET request. The market now includes Org1 and Org2 in the participants list, Org1 in the auctioneers list and Org2 in the Bidders list. | | |
| Input data | <ul style="list-style-type: none">Step 1 HTTP POST request with empty request bodyStep 2 { "ID": "8680ccf55f5c4d03845ee82d4b1c35e5", "PriorityTable": "6687732881c54b60a0168aefd4c6cf60", "Role": "Auctioneer" }Step 3 & Step 4 { "ID": "8680ccf55f5c4d03845ee82d4b1c35e5", "PriorityTable": "6687732881c54b60a0168aefd4c6cf60", | | |

| | |
|--|--|
| | <pre>"Role": "Bidder" }</pre> <ul style="list-style-type: none">• Step 5 <p>HTTP GET request without parameters</p> |
|--|--|

Result

• Step 1



The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** localhost:8090/initMarket
- Body Type:** raw (selected), JSON (application/json)
- Body Content:**

```

1 {
2   "Transaction ID":
3     "%!(EXTRA string=fa98f782ed48160f40d2a03ce167a4424530240cfbc1f9b96016ef2f15e60579)"
4 }

```
- Response:** Pretty, Raw, Preview, Auto (selected)
- Response Content:**

```

1 Transaction ID:
2 %!(EXTRA string=fa98f782ed48160f40d2a03ce167a4424530240cfbc1f9b96016ef2f15e60579)
3 )

```

• Step 2

POST

localhost:8091/applyToMarket

Params

Authorization

Headers (1)

Body

Pre-request Script

Tests

none

form-data

x-www-form-urlencoded

raw

binary

JSON (application/json)

```

1 {
2   "ID": "8680ccf55f5c4d03845ee82d4b1c35e5",
3   "PriorityTable": "6687732881c54b60a0168aefd4c6cf60",
4   "Role": "Auctioneer"
5 }

```

Body

Cookies

Headers (3)

Test Results

Pretty

Raw

Preview

Auto

```

1 Transaction ID:
2 %!(EXTRA string=3dea6fee545e617c7cfd09ce782502b91aa73a9e0f141b1879e3c8ad732e93f1
3 )

```

- Step 3

POST

localhost:8092/applyToMarket

Params

Authorization

Headers (1)

Body

Pre-request Script

Tests

none

form-data

x-www-form-urlencoded

raw

binary

JSON (application/json)

```

1 {
2   "ID": "8680ccf55f5c4d03845ee82d4b1c35e5",
3   "PriorityTable": "6687732881c54b60a0168aefd4c6cf60",
4   "Role": "Bidder"
5 }

```

Body

Cookies

Headers (3)

Test Results

Pretty

Raw

Preview

Auto

```

1 Transaction ID:
2 %!(EXTRA string=264d3fd8b00c32cfb4e8b5df2a5f2a3b9811a6416a908f70eaadb6f7469db1d4
3 )

```

- Step 4

POST

localhost:8093/applyToMarket

Params

Authorization

Headers (1)

Body

Pre-request Script

Tests

none

form-data

x-www-form-urlencoded

raw

binary

JSON (application/json)

```

1 {
2   "ID": "8680ccf55f5c4d03845ee82d4b1c35e5",
3   "PriorityTable": "6687732881c54b60a0168aefd4c6cf60",
4   "Role": "Bidder"
5 }

```

Body

Cookies

Headers (4)

Test Results

Pretty

Raw

Preview

Auto

```

1 Unable to invoke apply in market chaincode
2
3

```

- Step 5

Status: 200 OK Time: 10 ms Size: 1.15 KB Save

```

{"Participants":[{"MspId":"Org1MSP","CertId":
jY5SvncxLnV4YW1wbGUuY29tLE89b3JnM5S1eGfTccGxLLmVbSxMPVnNhb1BGcGFuY2l2Y28uU1Q9Q2FsaWZvcnSpYXN0PVVT"},{"MspId":"Org2MSP","CertId":
jY5SvncxLnV4YW1wbGUuY29tLE89b3JnM5S1eGfTccGxLLmVbSxMPVnNhb1BGcGFuY2l2Y28uU1Q9Q2FsaWZvcnSpYXN0PVVT"}], "Bidders":[{"MspId":"Org2MSP","CertId":
jY5SvncxLnV4YW1wbGUuY29tLE89b3JnM5S1eGfTccGxLLmVbSxMPVnNhb1BGcGFuY2l2Y28uU1Q9Q2FsaWZvcnSpYXN0PVVT"}], "Auctioneers":[{"MspId":"Org1MSP","CertId":
jY5SvncxLnV4YW1wbGUuY29tLE89b3JnM5S1eGfTccGxLLmVbSxMPVnNhb1BGcGFuY2l2Y28uU1Q9Q2FsaWZvcnSpYXN0PVVT"}], "Auctions":{"null":{}}

```

| | |
|------------------|----------|
| Test Case Result | Achieved |
|------------------|----------|

Table 32: E-AUCTION_04 unit test

| | | | |
|------------------|---|-----------|--|
| Test Case ID | E-AUCTION_04 | Component | E-auction module of energy trading framework |
| Description | This unit test showcases an end-to-end VCG auction procedure with 2 winners. Org1 is the auctioneer and 2 bidders apply to the market, Org2 and Org3. | | |
| Req ID | FR-UC4-01, FR-UC4-02, FR-UC4-07 | Priority | High |
| Prepared by | CERTH | Tested by | CERTH |
| Pre-condition(s) | <ul style="list-style-type: none">Each component of the e-auction has to be up and listening on their respective port, which includes at least an orderer, a peer for the ESCO organization, one or more peers for prosumers and a couchdb for each peer.The Market, Priority Table, ERC20 and Vickrey Clarke-Groves chaincodes must be installed and instantiated on the e-auction channel of the blockchain network.The application of each organization is runningThe priority table has been initialized and the ID is known to the organizations.A market has been initialized and the ID is known to the organizations. Org1 has applied as auctioneer, while Org2 and Org3 have applied as bidders.Org1, Org2 and Org3 have 500 tokens in their accounts.Org1, Org2 and Org3 have 0 initial market contribution. | | |
| Test steps | | | |
| 1 | Org1 (auctioneer) peer initiates a new Vickrey Clarke-Groves auction, selling 40KWh of energy with reserve price 5 tokens/KWh | | |
| 2 | Org1 peer queries blockchain to get the state of the auction with an HTTP GET request. The state should include the information of the Vickrey Clarke-Groves auction that Org1 provided as initialization parameters and 'State' field should be equal to 'Initialized'. | | |
| 3 | Org2 (bidder) peer submits a bid of 100 tokens for 15KWh of energy. | | |
| 4 | Org3 (bidder) peer submits a bid of 150 tokens for 20KWh of energy. | | |
| 5 | Org1 peer queries blockchain to get the state of the auction with an HTTP GET request. The state should include the cryptographic commitments to the bids of Org2 and Org3 and 'State' field should be equal to 'Bidding'. | | |

| | |
|----|---|
| 6 | Org1 peer queries the blockchain to retrieve the markets list with an HTTP GET request. The bidders list of the only existing market should be empty, as after each bidding the bidder is removed from the list of bidders. |
| 7 | Org2 peer reveals its bid to the Vickrey Clarke-Groves chaincode with an HTTP POST request. |
| 8 | Org3 peer reveals its bid to the Vickrey Clarke-Groves chaincode with an HTTP POST request. |
| 9 | Org1 peer queries blockchain to get the state of the auction with an HTTP GET request. The state should include the revealed bids of Org2 and Org3 and 'State' field should be equal to 'Revealing'. |
| 10 | Org1 peer awards winners with an HTTP PORT request. At this step the chaincode calculates the e-auction results. |
| 11 | Org1 peer queries blockchain to get the state of the auction with an HTTP GET request. The state should include the results of the e-auction, including the winners, the amount of energy sold, the amount of energy that was not sold, the shares of energy between winners and the amounts of tokens that they should pay to the auctioneer. The 'State' field should be equal to 'Awarding'. |
| 12 | Org1 peer queries the blockchain to retrieve the markets list with an HTTP GET request. The auctioneers list of the only existing market should be empty, as after awarding the auctioneer is removed from the list of auctioneers. |
| 13 | Org2 peer transfers the number of tokens that correspond to its bid to Org1's account with an HTTP POST request. |
| 14 | Org3 peer transfers the number of tokens that correspond to its bid to Org1's account with an HTTP POST request. |
| 15 | Org1 peer queries blockchain to get the state of the auction with an HTTP GET request. The 'State' field should be equal to 'Settling'. |
| 16 | Org1 peer retrieves its account balance through an HTTP GET request and should have 750 tokens remaining, after receiving 100 tokens from Org2 and 150 tokens from Org3. |
| 17 | Org2 peer retrieves its account balance through an HTTP GET request and should have 400 tokens remaining after transferring 100 tokens to Org1's account. |
| 18 | Org3 peer retrieves its account balance through an HTTP GET request and should have 350 tokens remaining after transferring 150 tokens to Org1. |
| 19 | Org1 peer retrieves the priority table through an HTTP GET request. Org1's contribution should be 35KWh, Org2's contribution should be equal to 15KWh and Org3's contribution should be equal to 20KWh. |

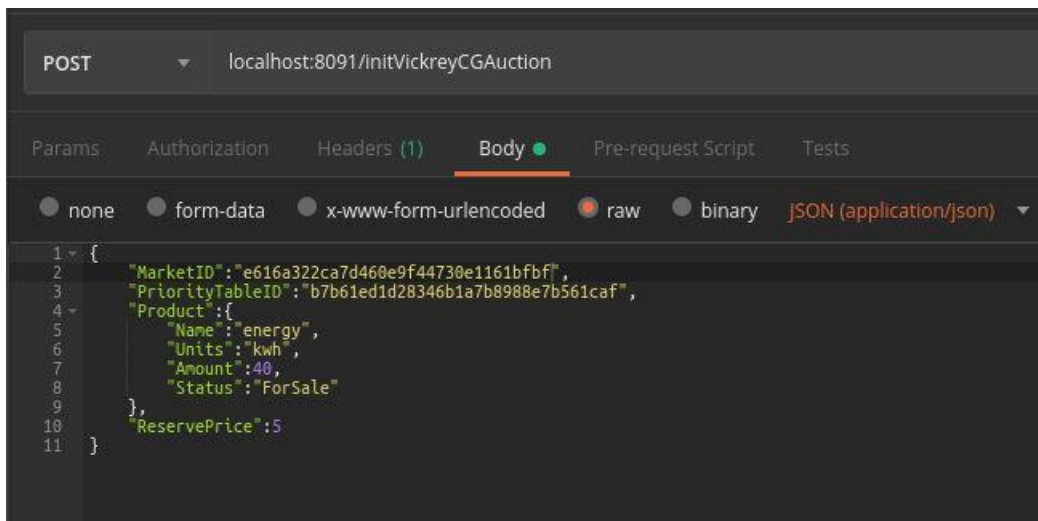
| | |
|------------|---|
| Input data | <ul style="list-style-type: none"> Step 1 <pre>{ "MarketID":"e616a322ca7d460e9f44730e1161bfbf", "PriorityTableID":"b7b61ed1d28346b1a7b8988e7b561caf", "Product":{ "Name":"energy", "Units":"kwh", "Amount":40, "Status":"ForSale" }, "ReservePrice":5 }</pre> <ul style="list-style-type: none"> Step 2 <p>auctionNum=5f57ce336f9a4962a3e2a32b5fff93e5</p> <ul style="list-style-type: none"> Step 3 <pre>{ "AuctionNum":"5f57ce336f9a4962a3e2a32b5fff93e5", "Bid": 100, "Amount": 15 }</pre> <ul style="list-style-type: none"> Step 4 <pre>{ "AuctionNum":"5f57ce336f9a4962a3e2a32b5fff93e5", "Bid": 150, "Amount": 20 }</pre> <ul style="list-style-type: none"> Step 5 <p>auctionNum=5f57ce336f9a4962a3e2a32b5fff93e5</p> <ul style="list-style-type: none"> Step 6 <p>HTTP GET request without parameters</p> <ul style="list-style-type: none"> Step 7 <pre>{</pre> |
|------------|---|

| | |
|--|--|
| | <pre>"AuctionNum": "5f57ce336f9a4962a3e2a32b5fff93e5", "Bid": 100, "Amount": 15 }</pre> <ul style="list-style-type: none"> • Step 8 <pre>{ "AuctionNum": "5f57ce336f9a4962a3e2a32b5fff93e5", "Bid": 150, "Amount": 20 }</pre> <ul style="list-style-type: none"> • Step 9 <p>auctionNum=5f57ce336f9a4962a3e2a32b5fff93e5</p> <ul style="list-style-type: none"> • Step 10 <pre>{ "AuctionNum": "5f57ce336f9a4962a3e2a32b5fff93e5" }</pre> <ul style="list-style-type: none"> • Step 11 <p>auctionNum=5f57ce336f9a4962a3e2a32b5fff93e5</p> <ul style="list-style-type: none"> • Step12 <p>HTTP GET request without parameters</p> <ul style="list-style-type: none"> • Step13 &Step 14 <pre>{ "AuctionNum": "5f57ce336f9a4962a3e2a32b5fff93e5" }</pre> <ul style="list-style-type: none"> • Step15 <p>auctionNum=5f57ce336f9a4962a3e2a32b5fff93e5</p> <ul style="list-style-type: none"> • Step16 <p>name=tok symbol=BT</p> |
|--|--|

| | |
|--|--|
| | <p>mspID=Org1MSP certID=eDU...VVT</p> <ul style="list-style-type: none">• Step17 name=tok symbol=BT mspID=Org2MSP certID=eDU...VVT• Step18 name=tok symbol=BT mspID=Org3MSP certID=eDU...VVT• Step19 HTTP GET request without parameters |
|--|--|

Result

• Step 1



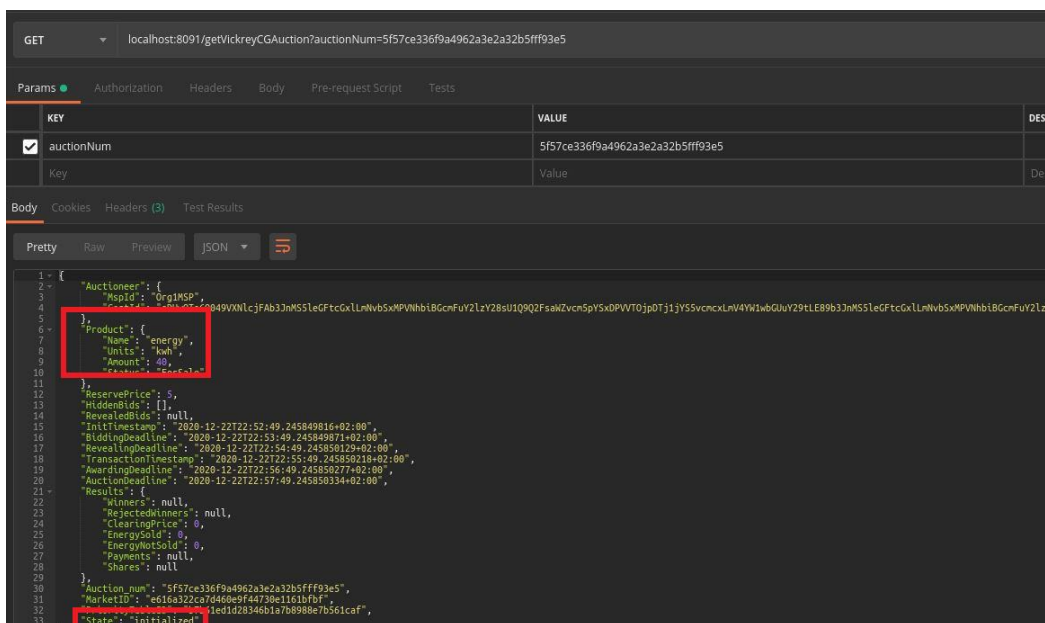
```

POST localhost:8091/initVickreyCGAuction

{
  "MarketID": "e616a322ca7d460e9f44730e1161bfbf",
  "PriorityTableID": "b7b61ed1d28346b1a7b8988e7b561caf",
  "Product": {
    "Name": "energy",
    "Units": "kwh",
    "Amount": 40,
    "Status": "ForSale"
  },
  "ReservePrice": 5
}

```

• Step 2



```

GET localhost:8091/getVickreyCGAuction?auctionNum=5f57ce336f9a4962a3e2a32b5fff93e5

{
  "Auctioneer": {
    "MspId": "Org1MSP",
    "OrgName": "Org1"
  },
  "Product": {
    "Name": "energy",
    "Units": "kwh",
    "Amount": 40,
    "Status": "ForSale"
  },
  "ReservePrice": 5,
  "HiddenBids": [],
  "RevealedBids": null,
  "InitTimestamp": "2020-12-22T22:52:49.245849816+02:00",
  "BiddingDeadline": "2020-12-22T22:53:49.245849871+02:00",
  "RevealingDeadline": "2020-12-22T22:54:49.245850129+02:00",
  "TransactionTimestamp": "2020-12-22T22:55:49.245850218+02:00",
  "AuctioningDeadline": "2020-12-22T22:56:49.245850277+02:00",
  "AuctionDeadline": "2020-12-22T22:57:49.245850334+02:00",
  "Results": {
    "Winners": null,
    "RejectedWinners": null,
    "ClearingPrice": 0,
    "EnergySold": 0,
    "EnergyNotSold": 0,
    "Payments": null,
    "Shares": null
  },
  "AuctionNum": "5f57ce336f9a4962a3e2a32b5fff93e5",
  "MarketID": "e616a322ca7d460e9f44730e1161bfbf",
  "PriorityTableID": "b7b61ed1d28346b1a7b8988e7b561caf",
  "State": "initialized"
}

```

• Step 3

POST

localhost:8092/VCGsubmitSealedBid

Params

Authorization

Headers (1)

Body

Pre-request Script

Tests

none

form-data

x-www-form-urlencoded

raw

binary

JSON (application/json)

```

1 {
2   "AuctionNum": "5f57ce336f9a4962a3e2a32b5fff93e5",
3   "Bid": 100,
4   "Amount": 15
5 }

```

Body

Cookies

Headers (3)

Test Results

Pretty

Raw

Preview

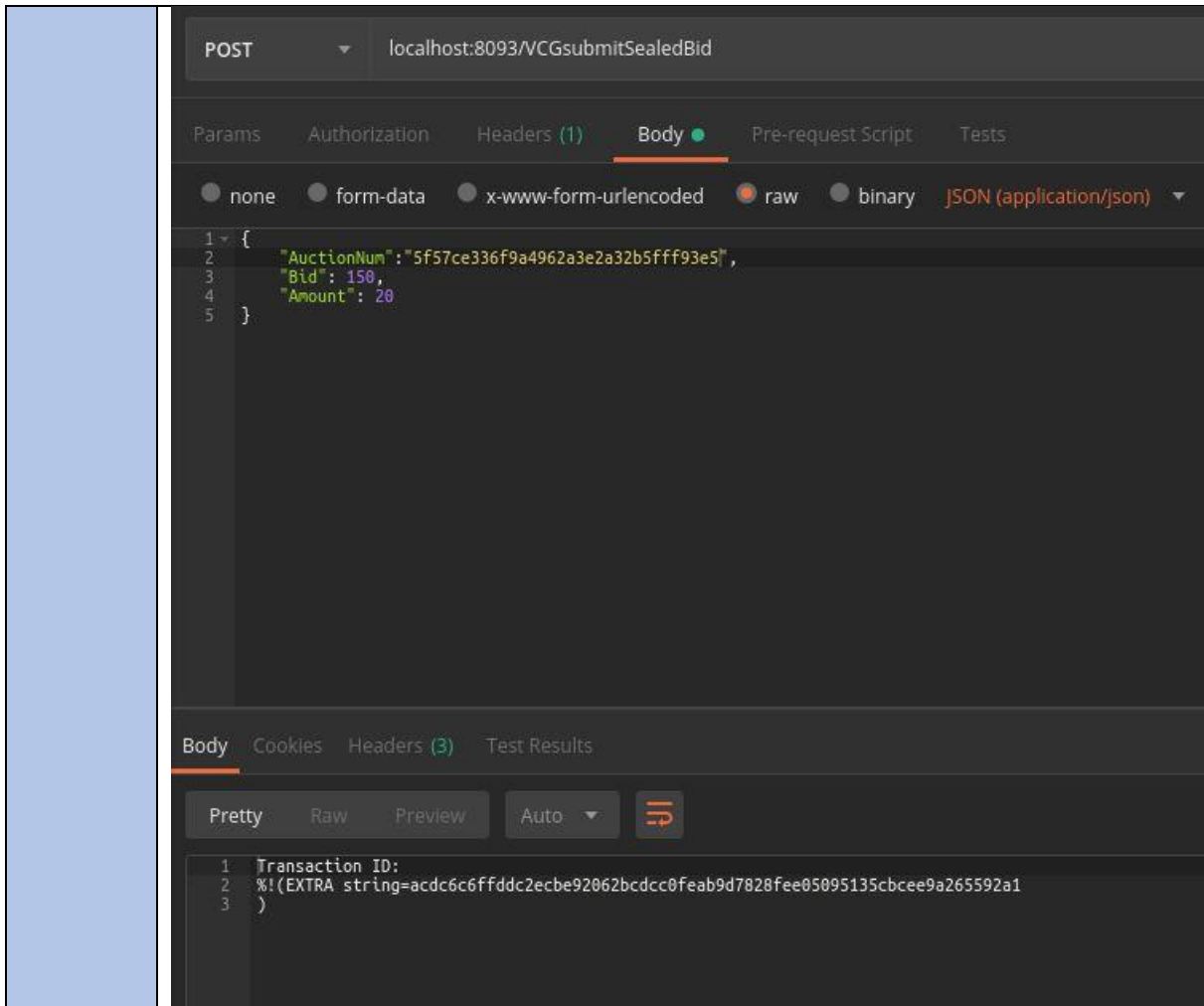
Auto

```

1 Transaction ID:
2 %!(EXTRA string=519a955d6a23ab6496bfb018e76627892fa5f78de77a5881a4aa2a8a96b77c0e
3 )

```

- Step 4



The screenshot displays a REST client interface. The top bar shows a POST request to `localhost:8093/VCGsubmitSealedBid`. The 'Body' tab is selected, showing a JSON payload:

```
1 {
2   "AuctionNum": "5f57ce336f9a4962a3e2a32b5fff93e5",
3   "Bid": 150,
4   "Amount": 20
5 }
```

Below the body, the 'Test Results' tab is active, showing the response body in 'Pretty' format:

```
1 Transaction ID:
2 %!(EXTRA string=acdc6c6ffddc2ecbe92062bcdcc0feab9d7828fee05095135cbcee9a265592a1
3 )
```

- **Step 5**

[illegible]

- **Step 6**

[illegible]

- **Step 7**

POST

localhost:8092/VCGreveal

Params

Authorization

Headers (1)

Body

Pre-request Script

Tests

none

form-data

x-www-form-urlencoded

raw

binary

JSON (application/json)

```

1 {
2   "AuctionNum": "5f57ce336f9a4962a3e2a32b5fff93e5",
3   "Bid": 100,
4   "Amount": 15
5 }
```

Body

Cookies

Headers (3)

Test Results

Pretty

Raw

Preview

Auto

```

1 Transaction ID:
2 %!(EXTRA string=ab198c24df4a5078fc3a90dd3f5d5a4dc8398f06b9a8091d3a779dc33b35cfa4
3 )
```

- Step 8

POST

localhost:8093/VCGreveal

Params

Authorization

Headers (1)

Body

Pre-request Script

Tests

none

form-data

x-www-form-urlencoded

raw

binary

JSON (application/json)

```

1 {
2   "AuctionNum": "5f57ce336f9a4962a3e2a32b5fff93e5",
3   "Bid": 150,
4   "Amount": 20
5 }

```

Body

Cookies

Headers (3)

Test Results

Pretty

Raw

Preview

Auto

```

1 Transaction ID:
2 %!(EXTRA string=27cee71bb8a02bfc20279d20d9f5aaffb7199e012b7369043c8230a9d2826fd
3 )

```

- Step 9

GET localhost:8091/getVikreyCGAuction?auctionNum=5f57ce336f9a462a3e2a32b5ff93e5

Pretty Raw Preview JSON

```

1 {
2   "Auctioneer": {
3     "MspId": "Org1MSP",
4     "CertId": "e8Uw0Tob0q049VXNlcjFAB33mW51eGfCcGxLLmNvbSxMPVNBb1BGcnFuY21zY28uU1Q9Q2FsaWZvcnSpYScDPVVT0jPDtj1jY5SvncvLnW4YV1wbG9uY29LE89b3JnW51eGfCcGxLLmNvbSxMPVNBb1BGcnFuY21zY28uU1Q9Q2FsaWZvcn3
5
6   },
7   "Product": {
8     "Name": "energy",
9     "Units": "kwh",
10    "Amount": 10,
11    "Status": "ForSale"
12  },
13  "ReservePrice": 5,
14  "HiddenBids": [
15    {
16      "Bidder": {
17        "MspId": "Org2MSP",
18        "CertId": "e8Uw0Tob0q049VXNlcjFAB33mW51eGfCcGxLLmNvbSxMPVNBb1BGcnFuY21zY28uU1Q9Q2FsaWZvcnSpYScDPVVT0jPDtj1jY5SvncvLnW4YV1wbG9uY29LE89b3JnW51eGfCcGxLLmNvbSxMPVNBb1BGcnFuY21zY28uU1Q9Q2F
19        "Bid": "5MhYgoPRfgDe3JFmGr5fMcXCB8FTerbnBc4JC8Hk=",
20        "Nonce": "QrH82eLQ9vD1JvnlpXm="
21      },
22    },
23    {
24      "Bidder": {
25        "MspId": "Org3MSP",
26        "CertId": "e8Uw0Tob0q049VXNlcjFAB33mW51eGfCcGxLLmNvbSxMPVNBb1BGcnFuY21zY28uU1Q9Q2FsaWZvcnSpYScDPVVT0jPDtj1jY5SvncvLnW4YV1wbG9uY29LE89b3JnW51eGfCcGxLLmNvbSxMPVNBb1BGcnFuY21zY28uU1Q9Q2F
27        "Bid": "P8Q+Sk7WmWYduZdXG855eKx1N04+GzyKJ5ox3BzKew",
28        "Nonce": "pVusVqY35tDYOPAmynM3N="
29      },
30    },
31  ],
32  "RevealedBids": [
33    {
34      "Bidder": {
35        "MspId": "Org3MSP",
36        "CertId": "e8Uw0Tob0q049VXNlcjFAB33mW51eGfCcGxLLmNvbSxMPVNBb1BGcnFuY21zY28uU1Q9Q2FsaWZvcnSpYScDPVVT0jPDtj1jY5SvncvLnW4YV1wbG9uY29LE89b3JnW51eGfCcGxLLmNvbSxMPVNBb1BGcnFuY21zY28uU1Q9Q2F
37        "Bid": 150,
38        "Amount": 20
39      },
40    },
41    {
42      "Bidder": {
43        "MspId": "Org2MSP",
44        "CertId": "e8Uw0Tob0q049VXNlcjFAB33mW51eGfCcGxLLmNvbSxMPVNBb1BGcnFuY21zY28uU1Q9Q2FsaWZvcnSpYScDPVVT0jPDtj1jY5SvncvLnW4YV1wbG9uY29LE89b3JnW51eGfCcGxLLmNvbSxMPVNBb1BGcnFuY21zY28uU1Q9Q2F
45        "Bid": 180,
46        "Amount": 15
47      },
48    },
49  ],
50  "InitTimestamp": "2020-12-22T22:55:49.245849816+02:00",
51  "BiddingDeadline": "2020-12-22T22:53:49.245849871+02:00",
52  "RevealingDeadline": "2020-12-22T22:54:49.245850129+02:00",
53  "TransactionTimestamp": "2020-12-22T22:55:49.245850218+02:00",
54  "AwardingDeadline": "2020-12-22T22:56:49.245850277+02:00",
55  "AuctionDeadline": "2020-12-22T22:57:49.245850334+02:00",
56  "Results": {
57    "Winners": null,
58    "RejectedWinners": null,
59    "ClearingPrice": 0,
60    "EnergySold": 0,
61    "EnergyNotSold": 0,
62    "Payments": null,
63    "Shares": null
64  },
65  "AuctionNum": "5f57ce336f9a462a3e2a32b5ff93e5",
66  "MarketID": "e516a325ca79469b9f4479be155bfa4",
67  "PrivateKeyID": "7b61ed1d28346b1a768988e7b561caf",
68  "State": "revealing"
69

```

- Step 10

© SDN microSENSE consortium
Public document

Page | 104

POST

localhost:8091/VCGaward

Params

Authorization

Headers (1)

Body

Pre-request Script

Tests

none

form-data

x-www-form-urlencoded

raw

binary

JSON (application/json)

```

1 {
2   "AuctionNum": "5f57ce336f9a4962a3e2a32b5fff93e5"
3 }

```

Body

Cookies

Headers (3)

Test Results

Pretty

Raw

Preview

Auto

```

1 Transaction ID:
2 %!(EXTRA string=2a8ba25b773dc9f329e4d69bae1e060b87e3f3196a49485ead4c37ec53722b4e
3 )

```

- Step11

GET
localhost:8091/getVikreyCGAuction?auctionNum=5f57ce336f9a4962a3e2a32b5ff93e5

Pretty
Raw
Preview
JSON

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83

```

- Step 12

Body
Cookies
Headers
Text Results
Status: 200 OK Time: 21 ms

Pretty
Raw
Preview
Auto

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83

```

- Step 13

POST

localhost:8092/VCGsettle

Params

Authorization

Headers (1)

Body

Pre-request Script

Tests

none

form-data

x-www-form-urlencoded

raw

binary

JSON (application/json)

```

1 {
2   "AuctionNum": "5f57ce336f9a4962a3e2a32b5fff93e5"
3 }

```

Body

Cookies

Headers (3)

Test Results

Pretty

Raw

Preview

Auto

```

1 Transaction ID:
2 %!(EXTRA string=47bddeea0bfe806b0e8af3f40e27bb1f4bd7779d1cda641b62900aa2666e2c8c
3 )

```

- Step14

POST

localhost:8093/VCGsettle

Params

Authorization

Headers (1)

Body

Pre-request Script

Tests

none

form-data

x-www-form-urlencoded

raw

binary

JSON (application/json)

```

1 {
2   "AuctionNum": "5f57ce336f9a4962a3e2a32b5fff93e5"
3 }

```

Body

Cookies

Headers (3)

Test Results

Pretty

Raw

Preview

Auto

```

1 Transaction ID:
2 %!(EXTRA string=ab91fc882a2c325b3d1e33a3dec68799a74df694bbfeb054c905789bbd00e0d7
3 )

```

- Step15

[illegible]

- **Step16**

GET

localhost:8091/getBalance?name=tok&symbol=BT&mspiD=Org1MSP&certID=eDUwOTo6Q049VXNlGjFAB3JnMS5leGFtcGxlLmNvbSxMPVNhbiBGcmFuZ2

Params

Authorization

Headers

Body

Pre-request Script

Tests

| KEY | VALUE |
|--|--|
| <input checked="" type="checkbox"/> name | tok |
| <input checked="" type="checkbox"/> symbol | BT |
| <input checked="" type="checkbox"/> mspiD | Org1MSP |
| <input checked="" type="checkbox"/> certID | eDUwOTo6Q049VXNlGjFAB3JnMS5leGFtcGxlLmNvbSxMPVNhbiBGcmFuZ2 |
| Key | Value |

Body

Cookies

Headers (3)

Test Results

Pretty

Raw

Preview

JSON

```
1 {
2   "Balance": "750"
3 }
```

- **Step17**

GET

localhost:8092/getBalance?name=tok&symbol=BT&mSPID=Org2MSP&certID=eDUwOTo6Q049VXNlqjFab3jnMi5leGFtcGxLmNvbSxMPVNhbiBGMcfUY2

Params

Authorization

Headers

Body

Pre-request Script

Tests

| KEY | VALUE |
|--|---|
| <input checked="" type="checkbox"/> name | tok |
| <input checked="" type="checkbox"/> symbol | BT |
| <input checked="" type="checkbox"/> mSPID | Org2MSP |
| <input checked="" type="checkbox"/> certID | eDUwOTo6Q049VXNlqjFab3jnMi5leGFtcGxLmNvbSxMPVNhbiBGMcfUY2 |
| Key | Value |

Body

Cookies

Headers (3)

Test Results

Pretty

Raw

Preview

JSON

1 {

2 "balance": "400"

3 }

- **Step18**

GET

localhost:8093/getBalance?name=tok&symbol=BT&mspID=Org3MSP&certID=eDUwOTo6Q049VXNlcjFAb3JnMy5leGFTcGxLmNvbSxMPVNhbIBGcmFuY2

Params

Authorization

Headers

Body

Pre-request Script

Tests

| KEY | VALUE |
|--|---|
| <input checked="" type="checkbox"/> name | tok |
| <input checked="" type="checkbox"/> symbol | BT |
| <input checked="" type="checkbox"/> mspID | Org3MSP |
| <input checked="" type="checkbox"/> certID | eDUwOTo6Q049VXNlcjFAb3JnMy5leGFTcGxLmNvbSxMPVNhbIBGcmFuY2 |
| Key | Value |

Body

Cookies

Headers (3)

Test Results

Pretty

Raw

Preview

JSON

```
1 {
2   "balance": "350"
3 }
```

- **Step19**

| Test Case | Result |
|-----------|--------|
| 1 | Pass |
| 2 | Pass |
| 3 | Pass |
| 4 | Pass |
| 5 | Pass |
| 6 | Pass |
| 7 | Pass |
| 8 | Pass |
| 9 | Pass |
| 10 | Pass |
| 11 | Pass |
| 12 | Pass |
| 13 | Pass |
| 14 | Pass |
| 15 | Pass |
| 16 | Pass |
| 17 | Pass |
| 18 | Pass |
| 19 | Pass |
| 20 | Pass |
| 21 | Pass |
| 22 | Pass |
| 23 | Pass |
| 24 | Pass |
| 25 | Pass |
| 26 | Pass |
| 27 | Pass |
| 28 | Pass |
| 29 | Pass |
| 30 | Pass |
| 31 | Pass |
| 32 | Pass |
| 33 | Pass |
| 34 | Pass |
| 35 | Pass |
| 36 | Pass |
| 37 | Pass |
| 38 | Pass |
| 39 | Pass |
| 40 | Pass |
| 41 | Pass |
| 42 | Pass |
| 43 | Pass |
| 44 | Pass |
| 45 | Pass |
| 46 | Pass |
| 47 | Pass |
| 48 | Pass |
| 49 | Pass |
| 50 | Pass |
| 51 | Pass |
| 52 | Pass |
| 53 | Pass |
| 54 | Pass |
| 55 | Pass |
| 56 | Pass |
| 57 | Pass |
| 58 | Pass |
| 59 | Pass |
| 60 | Pass |
| 61 | Pass |
| 62 | Pass |
| 63 | Pass |
| 64 | Pass |
| 65 | Pass |
| 66 | Pass |
| 67 | Pass |
| 68 | Pass |
| 69 | Pass |
| 70 | Pass |
| 71 | Pass |
| 72 | Pass |
| 73 | Pass |
| 74 | Pass |
| 75 | Pass |
| 76 | Pass |
| 77 | Pass |
| 78 | Pass |
| 79 | Pass |
| 80 | Pass |
| 81 | Pass |
| 82 | Pass |
| 83 | Pass |
| 84 | Pass |
| 85 | Pass |
| 86 | Pass |
| 87 | Pass |
| 88 | Pass |
| 89 | Pass |
| 90 | Pass |
| 91 | Pass |
| 92 | Pass |
| 93 | Pass |
| 94 | Pass |
| 95 | Pass |
| 96 | Pass |
| 97 | Pass |
| 98 | Pass |
| 99 | Pass |
| 100 | Pass |

Achieved

Table 33: E-AUCTION_05 unit test

| | | | |
|------------------|--|-----------|--|
| Test Case ID | E-AUCTION_05 | Component | E-auction module of energy trading framework |
| Description | This unit test showcases the rejection of a bidder that bids to an auction after the bidding deadline. | | |
| Req ID | FR-UC4-07 | Priority | Medium |
| Prepared by | CERTH | Tested by | CERTH |
| Pre-condition(s) | <ul style="list-style-type: none">Each component of the e-auction has to be up and listening on their respective port, which includes at least an orderer, a peer for the ESCO organization, one or more peers for prosumers and a couchdb for each peer.The Market, Priority Table, ERC20 and Vickrey Clarke-Groves chaincodes must be installed and instantiated on the e-auction channel of the blockchain network.The application of each organization is runningThe priority table has been initialized and the ID is known to the organizations.A market has been initialized and the ID is known to the organizations. Org1 has applied as auctioneer, while Org2 and Org3 have applied as bidders. | | |
| Test steps | | | |
| 1 | Org1 (auctioneer) peer initiates a new Vickrey Clarke-Groves auction, selling 40KWh of energy with reserve price 5 tokens/KWh | | |
| 2 | Org2 (bidder) peer submits a bid of 100 tokens for 15KWh of energy. | | |
| 3 | Org3 (bidder) peer submits a bid of 150 tokens for 20KWh of energy but the bid is overdue. | | |
| 4 | Org1 peer queries the blockchain to retrieve the markets list with an HTTP GET request. The bidders list of the only existing market should include Org3, since Org3's peer bid got rejected and it should have the opportunity to take part in a subsequent auction of the market. | | |
| Input data | <ul style="list-style-type: none">Step 1 <pre>{ "MarketID": "", "PriorityTableID": "", "Product": { "Name": "energy", "Units": "kwh", "Amount": 40, "Status": "ForSale" } },</pre> | | |

| | |
|--------|--|
| | <pre>"ReservePrice":5 }</pre> <ul style="list-style-type: none"> Step 2 <pre>{ "AuctionNum": "", "Bid": 100, "Amount": 15 }</pre> <ul style="list-style-type: none"> Step 3 <pre>{ "AuctionNum": "", "Bid": 150, "Amount": 20 }</pre> <ul style="list-style-type: none"> Step 4 <p>HTTP GET request without parameters</p> |
| Result | <ul style="list-style-type: none"> Step 1 |

POST

localhost:8091/initVickreyCGAuction

Params

Authorization

Headers (1)

Body

Pre-request Script

Tests

none

form-data

x-www-form-urlencoded

raw

binary

JSON (application/json)

```

1 {
2   "MarketID": "db6423445a34465e934ec37a30cd36af",
3   "PriorityTableID": "5de09e08b5fb418d82395d66cfe34bf3",
4   "Product": {
5     "Name": "energy",
6     "Units": "kwh",
7     "Amount": 40,
8     "Status": "ForSale"
9   },
10  "ReservePrice": 5
11 }

```

Body

Cookies

Headers (3)

Test Results

Pretty

Raw

Preview

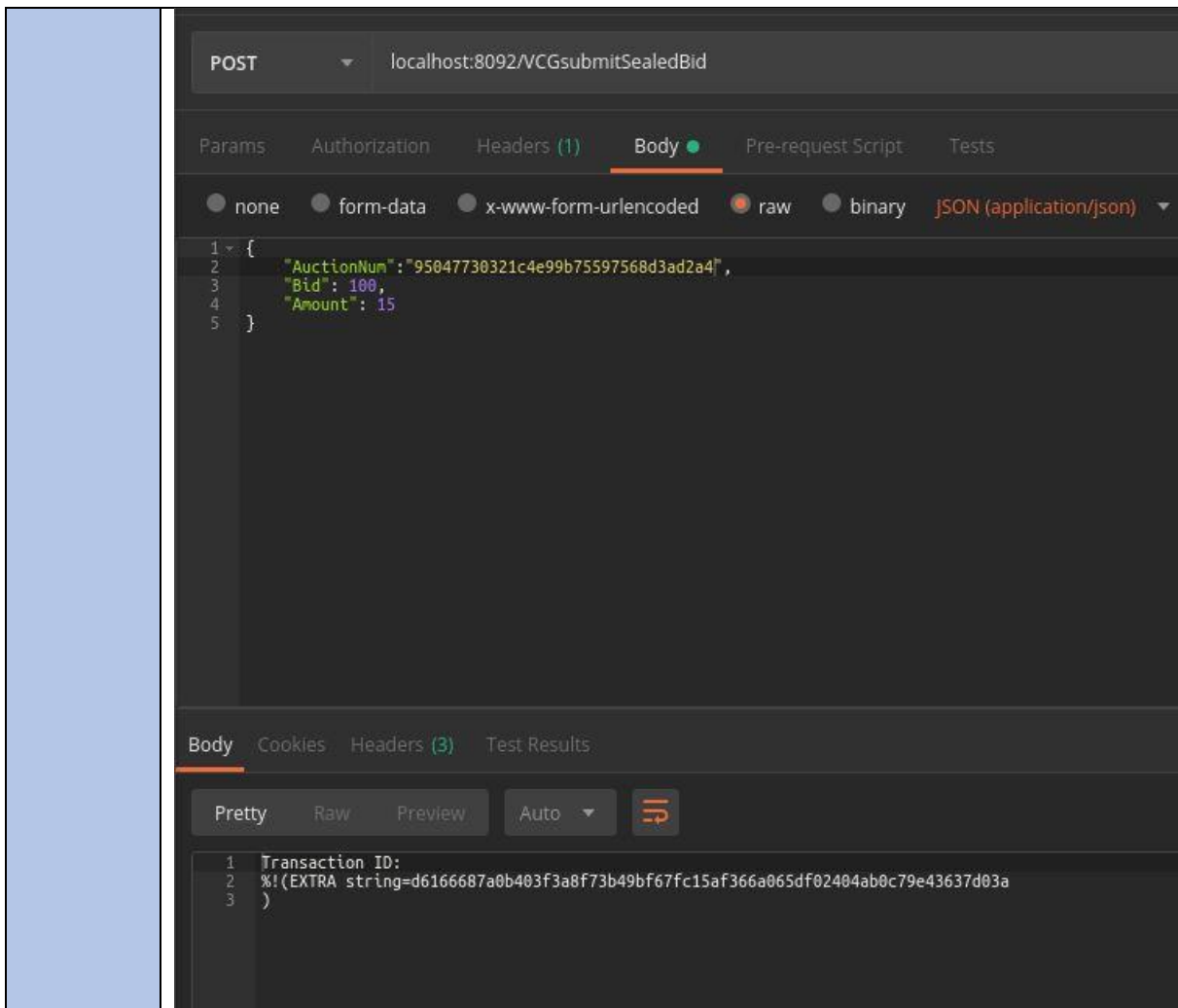
Auto

```

1 Transaction ID:
2 %!(EXTRA string=3387d3d949126363b19b4ec6bfd57febba356a2340491ec448b0640969ad2b1e
3 )

```

- Step 2



The screenshot shows a REST client interface. The top bar indicates a POST request to `localhost:8092/VCGsubmitSealedBid`. The 'Body' tab is selected, showing a JSON payload: `{ "AuctionNum": "95047730321c4e99b75597568d3ad2a4", "Bid": 100, "Amount": 15 }`. The 'Test Results' tab is also visible, showing a response with a Transaction ID: `Transaction ID: %!(EXTRA string=d6166687a0b403f3a8f73b49bf67fc15af366a065df02404ab0c79e43637d03a)`.

- **Step 3**

[illegible]

5.2 Blockchain Based Intrusion and Anomaly Detection module unit tests

Table 34: BIAD_01 unit test

| | | | |
|------------------|--|-----------|----------------------------|
| Test Case ID | BIAD_01 | Component | BIAD network and chaincode |
| Description | The device is registered in the blockchain to be identified for futures interactions. | | |
| Req ID | FR-GR-06; FR-UC4-05 | Priority | High |
| Prepared by | TECNALIA | Tested by | TECNALIA |
| Pre-condition(s) | <ul style="list-style-type: none">Each component of the BIAD has to be up and listening on their respective port, which includes at least an orderer (7050), a peer (7051), the CA (7054), couchdb (5084) and REST API (4040).The chaincode must be installed and instantiated on the blockchain network.The device must have a way to sign a transaction. In this case it is done via Rest API, so it must be log in. | | |
| Test steps | | | |
| 1 | The device sends a POST HTTP request to the API, using RegisterDevice method, with its IP as identifier in the body and the authentication token in header. | | |
| 2 | The orderer receives and mined the transaction in the next block, so that it can be added to the chain. | | |
| 3 | Check if the device has been correctly registered using GetDeviceInfo route at the API. | | |
| Input data | {“dvcID”:”10.10.10.100”} | | |
| Result | <ul style="list-style-type: none">Step 1 | | |

POST
cpd_url / channel / chaincode / invoke/registerDevice
Send

JSON
Bearer
Query
Header 1
Docs

```

1 {
2   "dvcID": "10.10.10.100"
3 }

```

Beautify JSON

200 OK
2.49 s
213 B
Just Now

Preview
Header 12
Cookie
Timeline

```

1 {
2   "type": "invoke",
3   "channelName": "ch1",
4   "chaincodeId": "ccsdn",
5   "fcn": "registerDevice",
6   "transactionID":
7     "73677669b04fa4348a59cb3fd4802602a487421626d6f03edc2b3fdec1a33e3f",
8   "result": {
9     "dvcID": "10.10.10.100",
10    "status": true
11  }

```

- Step 2

```

2020-11-27 10:24:53.561 UTC [dispatcher] Debug -> DEBU 785 invoke() called
2020-11-27 10:24:53.561 UTC [dispatcher] Debug -> DEBU 786 initializing ledgerbuilder internal pools
2020-11-27 10:24:53.561 UTC [dispatcher] Debug -> DEBU 787 runner stub runtime detected as: *fabric.ChaincodeStub
2020-11-27 10:24:53.562 UTC [dispatcher] Debug -> DEBU 788 initializing ledgerbuilder internal pools for production environment
2020-11-27 10:24:53.562 UTC [dispatcher] Debug -> DEBU 789 creating new smart stub for hyperledger fabric
2020-11-27 10:24:53.562 UTC [dispatcher] Debug -> DEBU 78a fetching requested operation in from current chaincode runner...
2020-11-27 10:24:53.562 UTC [dispatcher] Info -> INFO 78b getting current operation id from stub encoded data
2020-11-27 10:24:53.562 UTC [dispatcher] Info -> INFO 78c received argument count is: 2
2020-11-27 10:24:53.562 UTC [dispatcher] Info -> INFO 78d Requested operation key: registerDevice
2020-11-27 10:24:53.562 UTC [dispatcher] Info -> INFO 78e decoding chaincode stub payload content
2020-11-27 10:24:53.562 UTC [dispatcher] Info -> INFO 78f Received data (via stub) length: 24
2020-11-27 10:24:53.562 UTC [dispatcher] Info -> INFO 790 Received data (via stub) data is: {"dvcID":"10.10.10.100"}
2020-11-27 10:24:53.562 UTC [dispatcher] Info -> INFO 791 Received chaincode op key: registerDevice
2020-11-27 10:24:53.563 UTC [dispatcher] Debug -> DEBU 792 using operation stub decoder...
2020-11-27 10:24:53.563 UTC [controller] Debug -> DEBU 793 decoding stub using controller stub reader...
2020-11-27 10:24:53.563 UTC [controller] Debug -> DEBU 794 decoding stub using controller data structure...
2020-11-27 10:24:53.563 UTC [ccsdn] Debug -> DEBU 795 creating new device
2020-11-27 10:24:53.563 UTC [dispatcher] Info -> INFO 796 Parsed data from stub is: {"dvcID":"10.10.10.100","status":true}
2020-11-27 10:24:53.563 UTC [dispatcher] Debugf -> DEBU 797 detected data is of type: *model.DeviceModel
2020-11-27 10:24:53.563 UTC [dispatcher] Debug -> DEBU 798 Applying data preprocessing middlewares
2020-11-27 10:24:53.563 UTC [middleware] Info -> INFO 799 Checking asset against schema index = 0 middleware = request-not-nil
2020-11-27 10:24:53.564 UTC [ccsdn] Info -> INFO 79a Checking asset is not nil or empty
2020-11-27 10:24:53.564 UTC [middleware] Info -> INFO 79b Checking asset against schema index = 1 middleware = device-is-not-unique
2020-11-27 10:24:53.564 UTC [ccsdn] Notice -> NOTI 79c Checking if device 10.10.10.100 is unique
2020-11-27 10:24:53.569 UTC [ccsdn] Notice -> NOTI 79d Registering a new device
2020-11-27 10:24:53.570 UTC [controller] Info -> INFO 79e Saving asset with ID : *DeviceModel.10.10.10.100
2020-11-27 10:24:53.570 UTC [controller] Info -> INFO 79f Saving asset with DATA : {"dvcID":"10.10.10.100","status":true}
2020-11-27 10:24:53.571 UTC [controller] Debug -> DEBU 79a emitting put state event
2020-11-27 10:24:53.571 UTC [dispatcher] Info -> INFO 7a1 Returning result to client via stub 200 {"dvcID":"10.10.10.100","status":true}

```

- Step 3

| | |
|--|--|
| | <div> <div>POST ▾</div> <div>cpd_url / channel / chaincode /query/getDeviceInfo</div> <div>Send</div> </div> <div> <div>JSON ▾</div> <div>Bearer ▾</div> <div>Query</div> <div>Header 1</div> <div>Docs</div> </div> <div> <pre> 1 { 2 "dvcID": "10.10.10.100" 3 } </pre> <div>Beautify JSON</div> </div> <div> <div>200 OK</div> <div>267 ms</div> <div>128 B</div> <div>Just Now ▾</div> </div> <div> <div>Preview ▾</div> <div>Header 12</div> <div>Cookie</div> <div>Timeline</div> </div> <div> <pre> 1 { 2 "type": "query", 3 "channelName": "ch1", 4 "chaincodeId": "ccsdn", 5 "fcn": "getDeviceInfo", 6 "result": { 7 "dvcID": "10.10.10.100", 8 "status": true 9 } 10 } </pre> </div> |
| | <div>Test Case Result</div> <div>Achieved</div> |

Table 35: BIAD_02 unit test

| Test Case ID | BIAD_02 | Component | BIAD network and chaincode |
|------------------|--|-----------|----------------------------|
| Description | The device registers a hash and a processing parameter to be monitored with some initial values. If they already exist, the value is updated, and its validity is checked (same than BIAD_03 and BIAD_04 functionalities). | | |
| Req ID | FR-UR-06; FR-GR-06; FR-UC4-05 | Priority | High |
| Prepared by | TECNALIA | Tested by | TECNALIA |
| Pre-condition(s) | <ul style="list-style-type: none"> Each component of the BIAD has to be up and listening on their respective port, which includes at least an orderer (7050), a peer (7051), the CA (7054), couchdb (5084) and REST API (4040). | | |

| | |
|------------|--|
| | <ul style="list-style-type: none"> The chaincode must be installed and instantiated on the blockchain network. The device must have a way to sign a transaction. In this case it is done via Rest API, so it must be log in. The associated device must be previously registered on the blockchain and have a devID identifier. |
| Test steps | |
| 1 | The device sends a POST HTTP request to the API, using RegisterHash or RegisterRecord methods, with their respective initial values in the body in JSON format. |
| 2 | The orderer receives and mined the transaction in the next block, so that it can be added to the chain. |
| 3 | Check if the Hash/Record has been correctly registered using GetDeviceInfo route at the API. |
| Input data | RegisterHash: <pre>{ "dvcID": "10.10.10.100", "path": "/etc/passwd", "value": "fn3y09bty340c0rf354tv4635bv37c3f04n5v2v5204ocweoi"} </pre> RegisterRecord: <pre>{ "dvcID": "10.10.10.100", "param": "USED_MEM", "value": 5, "max": 60, "min": 2} </pre> |
| Result | <ul style="list-style-type: none"> Step 1 |

POST
cpd_url / channel / chaincode /invoke/registerHash
Send

JSON
Bearer
Query
Header 1
Docs

```

1 {
2   "dvcID": "10.10.10.100",
3   "path": "/etc/passwd",
4   "value": "fn3y09bty340c0rf354tv4635bv37c3f04n5v2v5204ocweoi"
5 }

```

Beautify JSON

200 OK
2.36 s
317 B
Just Now

Preview
Header 12
Cookie
Timeline

```

1 {
2   "type": "invoke",
3   "channelName": "ch1",
4   "chaincodeId": "ccsdn",
5   "fcn": "registerHash",
6   "transactionID":
   "c8d7bd74ffb5102ee88ec62e751b7c5c4106a67c9e512c6677dc5b7e28149963",
7   "result": {
8     "hashID": "1kryIbFfcxraifnJxt84I3kdGKJ",
9     "dvcID": "10.10.10.100",
10    "path": "/etc/passwd",
11    "value": "fn3y09bty340c0rf354tv4635bv37c3f04n5v2v5204ocweoi"
12  }
13 }

```

- Step 2


```

2020-11-27 10:28:45.601 UTC [dispatcher] Debug -> DEBU 7bc invoke() called
2020-11-27 10:28:45.602 UTC [dispatcher] Debug -> DEBU 7bd initializing ledgerbuilder internal pools
2020-11-27 10:28:45.602 UTC [dispatcher] Debug -> DEBU 7be runner stub runtime detected as: "fabric.ChaincodeStub
2020-11-27 10:28:45.602 UTC [dispatcher] Debug -> DEBU 7bf initializing ledgerbuilder internal pools for production environment
2020-11-27 10:28:45.603 UTC [dispatcher] Debug -> DEBU 7c0 creating new smart stub for hyperledger fabric
2020-11-27 10:28:45.603 UTC [dispatcher] Debug -> DEBU 7c1 fetching requested operation in from current chaincode runner...
2020-11-27 10:28:45.603 UTC [dispatcher] Info -> INFO 7c2 getting current operation id from stub encoded data
2020-11-27 10:28:45.604 UTC [dispatcher] Info -> INFO 7c3 received argument count is: 2
2020-11-27 10:28:45.604 UTC [dispatcher] Info -> INFO 7c4 Requested operation key: registerHash
2020-11-27 10:28:45.604 UTC [dispatcher] Info -> INFO 7c5 decoding chaincode stub payload content
2020-11-27 10:28:45.604 UTC [dispatcher] Info -> INFO 7c6 Received data (via stub) length: 105
2020-11-27 10:28:45.604 UTC [dispatcher] Info -> INFO 7c7 Received data (via stub) data is: {"dvcID":"10.10.10.100","path":"/etc/passwd","v
alue":"fn3y09bty340c0rf354tv4635bv37c3f04n5v2v5204ocweoi"}
2020-11-27 10:28:45.605 UTC [dispatcher] Info -> INFO 7c8 Received chaincode op key: registerHash
2020-11-27 10:28:45.605 UTC [dispatcher] Debug -> DEBU 7c9 using operation stub decoder...
2020-11-27 10:28:45.605 UTC [controller] Debug -> DEBU 7ca decoding stub using controller stub reader...
2020-11-27 10:28:45.605 UTC [controller] Debug -> DEBU 7cb decoding stub using controller data structure...
2020-11-27 10:28:45.605 UTC [ccsdn] Debug -> DEBU 7cc creating new hash
2020-11-27 10:28:45.606 UTC [dispatcher] Info -> INFO 7cd Parsed data from stub is: {"hashID":"1kryIbFfcxraifnJxt84I3kdGKJ","dvcID":"10.10.
10.100","path":"/etc/passwd","value":"fn3y09bty340c0rf354tv4635bv37c3f04n5v2v5204ocweoi"}
2020-11-27 10:28:45.606 UTC [dispatcher] Debug -> DEBU 7ce detected data is of type: "model.HashModel
2020-11-27 10:28:45.606 UTC [dispatcher] Debug -> DEBU 7cf Applying data preprocessing middlewares
2020-11-27 10:28:45.607 UTC [middleware] Info -> INFO 7d0 Checking asset against schema index = 0 middleware = hash-input-ok
2020-11-27 10:28:45.607 UTC [ccsdn] Info -> INFO 7d1 Checking hash inputs are ok
2020-11-27 10:28:45.607 UTC [middleware] Info -> INFO 7d2 Checking asset against schema index = 1 middleware = device-must-exist
2020-11-27 10:28:45.607 UTC [ccsdn] Log -> DEBU 7d3 Checking if device 10.10.10.100 exists
2020-11-27 10:28:45.613 UTC [ccsdn] Notice -> NOTI 7d4 Registering a new hash
2020-11-27 10:28:45.613 UTC [controller] Debug -> DEBU 7d5 Reading asset with ID : "DeviceModel.10.10.10.100
2020-11-27 10:28:45.616 UTC [controller] Debug -> DEBU 7d6 Readed data length: 38
2020-11-27 10:28:45.616 UTC [controller] Debug -> DEBU 7d7 Readed raw data is: {"dvcID":"10.10.10.100","status":true}
2020-11-27 10:28:45.617 UTC [ccsdn] Info -> INFO 7d8 Updating device
2020-11-27 10:28:45.617 UTC [controller] Debug -> DEBU 7d9 key split algorithm: disabled
2020-11-27 10:28:45.617 UTC [controller] Info -> INFO 7da Saving asset with ID : "DeviceModel.10.10.10.100
2020-11-27 10:28:45.618 UTC [controller] Info -> INFO 7db Saving asset with DATA : {"dvcID":"10.10.10.100","hashes":[{"hashID":"1kryIbFfcx
raifnJxt84I3kdGKJ","dvcID":"10.10.10.100","path":"/etc/passwd","value":"fn3y09bty340c0rf354tv4635bv37c3f04n5v2v5204ocweoi"}],"status":true}
2020-11-27 10:28:45.618 UTC [controller] Debug -> DEBU 7dc emitting put state event
2020-11-27 10:28:45.620 UTC [dispatcher] Info -> INFO 7dd Returning result to client via stub 200 {"hashID":"1kryIbFfcxraifnJxt84I3kdGKJ","
dvcID":"10.10.10.100","path":"/etc/passwd","value":"fn3y09bty340c0rf354tv4635bv37c3f04n5v2v5204ocweoi"}

```

• Step 3

POST
cpd_ur1 / channel / chaincode /query/getDeviceInfo
Send

JSON
Bearer
Query
Header 1
Docs

1 {
2 "dvcID": "10.10.10.100"
3 }

Beautify JSON

200 OK
241 ms
284 B
Just Now

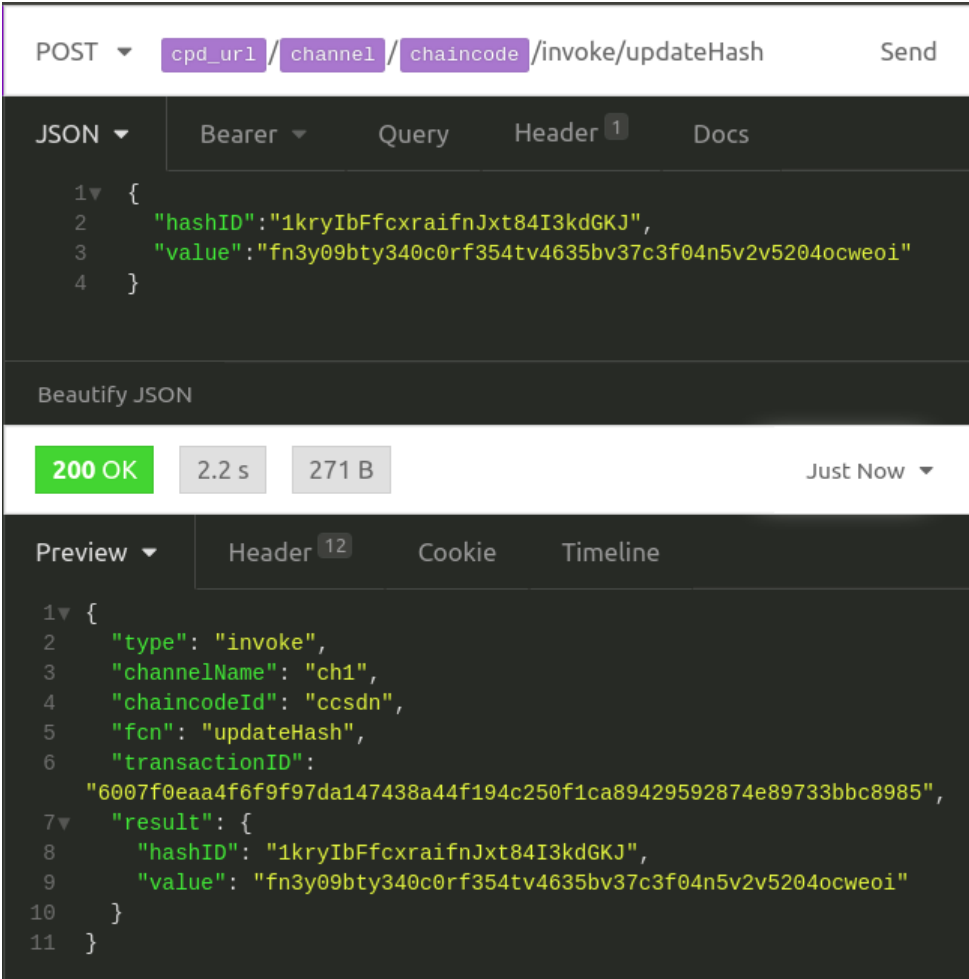
Preview
Header 12
Cookie
Timeline

1 {
2 "type": "query",
3 "channelName": "ch1",
4 "chaincodeId": "ccsdn",
5 "fcn": "getDeviceInfo",
6 "result": {
7 "dvcID": "10.10.10.100",
8 "hashes": [
9 {
10 "hashID": "1kryIbFfcxraifnJxt84I3kdGKJ",
11 "dvcID": "10.10.10.100",
12 "path": "/etc/passwd",
13 "value": "fn3y09bty340c0rf354tv4635bv37c3f04n5v2v5204ocweoi"
14 }
15],
16 "status": true
17 }
18 }

| | |
|------------------|----------|
| Test Case Result | Achieved |
|------------------|----------|

Table 36: BIAD_03 unit test

| | | | |
|------------------|--|-----------|----------------------------|
| Test Case ID | BIAD_03 | Component | BIAD network and chaincode |
| Description | The device updates the hash and the parameter registered on the blockchain, so it can be processed. This time there will not be any anomaly performance. This case is possible to do with <i>RegisterHash</i> function when the hash already exists on the ledger. | | |
| Req ID | FR-UR-06;FR-GR-06; FR-UC4-05 | Priority | High |
| Prepared by | TECNALIA | Tested by | TECNALIA |
| Pre-condition(s) | <ul style="list-style-type: none">Each component of the BIAD has to be up and listening on their respective port, which includes at least an orderer (7050), a peer (7051), the CA (7054), couchdb (5084) and REST API (4040).The chaincode must be installed and instantiated on the blockchain network.The device must have a way to sign a transaction. In this case it is done via Rest API, so it must be log in.The associated device must be previously registered on the blockchain and have a devID identifier. And in case of processing parameter, it must be previously registered as well. | | |
| Test steps | | | |
| 1 | The device sends a POST HTTP request to the API, using UpdateHash or UpdateRecord methods, with their respective identity and actual values in the body, in JSON format. | | |
| 2 | The orderer receives and mined the transaction in the next block, so that it can be added to the chain. | | |
| 3 | Check if the Hash/Record has been correctly updated and if there is any alert, using GetDeviceInfo route at the API. | | |
| Input data | UpdateHash: {"hashID":"1krylbFfcxraifnJxt84I3kdGKJ", "value":" fn3y09bty340c0rf354tv4635bv37c3f04n5v2v5204ocweoi"} | | |

| | |
|--------|---|
| | UpdateRecord: <pre>{ "recordID": "1kryIbFfcxraifnJxt84I3kdGKJ", "value": 6 }</pre> |
| Result | <ul style="list-style-type: none"> Step 1  <pre>POST cpd_url/channel/chaincode/invoke/updateHash</pre> <pre>{ "hashID": "1kryIbFfcxraifnJxt84I3kdGKJ", "value": "fn3y09bty340c0rf354tv4635bv37c3f04n5v2v5204ocweoi" }</pre> <pre>{ "type": "invoke", "channelName": "ch1", "chaincodeId": "ccsdn", "fcn": "updateHash", "transactionID": "6007f0eaa4f6f9f97da147438a44f194c250f1ca89429592874e89733bbc8985", "result": { "hashID": "1kryIbFfcxraifnJxt84I3kdGKJ", "value": "fn3y09bty340c0rf354tv4635bv37c3f04n5v2v5204ocweoi" } }</pre> <ul style="list-style-type: none"> Step 2 |

```

2020-11-27 10:41:08.029 UTC [dispatcher] Debug -> DEBU b6c invoke() called
2020-11-27 10:41:08.029 UTC [dispatcher] Debug -> DEBU b6d initializing ledgerbuildr internal pools
2020-11-27 10:41:08.030 UTC [dispatcher] Debug -> DEBU b6e runner stub runtime detected as: "fabric.ChaincodeStub
2020-11-27 10:41:08.030 UTC [dispatcher] Debug -> DEBU b6f initializing ledgerbuildr internal pools for production environment
2020-11-27 10:41:08.030 UTC [dispatcher] Debug -> DEBU b70 creating new smart stub for hyperledger fabric
2020-11-27 10:41:08.031 UTC [dispatcher] Debug -> DEBU b71 fetching requested operation in from current chaincode runner...
2020-11-27 10:41:08.031 UTC [dispatcher] Info -> INFO b72 getting current operation id from stub encoded data
2020-11-27 10:41:08.031 UTC [dispatcher] Info -> INFO b73 received argument count is: 2
2020-11-27 10:41:08.031 UTC [dispatcher] Info -> INFO b74 Requested operation key: updateHash
2020-11-27 10:41:08.032 UTC [dispatcher] Info -> INFO b75 decoding chaincode stub payload content
2020-11-27 10:41:08.032 UTC [dispatcher] Info -> INFO b76 Received data (via stub) length: 100
2020-11-27 10:41:08.032 UTC [dispatcher] Info -> INFO b77 Received data (via stub) data is: {"hashID":"1kryIbFfcxraifnJxt84I3kdGKJ","value":
"fn3y09bty340c0rf354tv4635bv37c3f04n5v2v5204ocweoi"}
2020-11-27 10:41:08.032 UTC [dispatcher] Info -> INFO b78 Received chaincode op key: updateHash
2020-11-27 10:41:08.033 UTC [dispatcher] Debug -> DEBU b79 using operation stub decoder...
2020-11-27 10:41:08.033 UTC [controller] Debug -> DEBU b7a decoding stub using controller stub reader...
2020-11-27 10:41:08.033 UTC [controller] Debug -> DEBU b7b decoding stub using controller data structure...
2020-11-27 10:41:08.033 UTC [ccsdn] Debug -> DEBU b7c creating new hash
2020-11-27 10:41:08.034 UTC [dispatcher] Info -> INFO b7d Parsed data from stub is: {"hashID":"1kryIbFfcxraifnJxt84I3kdGKJ","value":"fn3y09b
ty340c0rf354tv4635bv37c3f04n5v2v5204ocweoi"}
2020-11-27 10:41:08.034 UTC [dispatcher] Debugf -> DEBU b7e detected data is of type: "model.HashModel
2020-11-27 10:41:08.034 UTC [dispatcher] Debug -> DEBU b7f Applying data preprocessing middlewares
2020-11-27 10:41:08.034 UTC [middleware] Info -> INFO b80 Checking asset against schema index = 0 middleware = request-not-nil
2020-11-27 10:41:08.034 UTC [ccsdn] Info -> INFO b81 Checking asset is not nil or empty
2020-11-27 10:41:08.034 UTC [ccsdn] Info -> INFO b82 checking hashes
2020-11-27 10:41:08.039 UTC [controller] Debug -> DEBU b83 Reading asset with ID "DeviceModel.10.10.100
2020-11-27 10:41:08.042 UTC [controller] Debug -> DEBU b84 Readed data length: 104
2020-11-27 10:41:08.043 UTC [controller] Debug -> DEBU b85 Readed raw data is: {"dvcID":"10.10.10.100","hashes":[{"dvcID":"10.10.10.100","h
ashID":"1kryIbFfcxraifnJxt84I3kdGKJ","path":"/etc/passwd","value":"fn3y09bty340c0rf354tv4635bv37c3f04n5v2v5204ocweoi"}],"status":true}
2020-11-27 10:41:08.043 UTC [dispatcher] Info -> INFO b86 Returning result to client via stub 200 {"hashID":"1kryIbFfcxraifnJxt84I3kdGKJ","v
alue":"fn3y09bty340c0rf354tv4635bv37c3f04n5v2v5204ocweoi"}

```

Step 3

POST ▼ cpd_url / channel / chaincode / query/getDeviceInfo Send

JSON ▼ Bearer ▼ Query Header 1 Docs

```

1▼ {
2  "dvcID": "10.10.10.100"
3 }

```

Beautify JSON

200 OK

270 ms

284 B

Just Now ▼

Preview ▼ Header 12 Cookie Timeline

```

1▼ {
2  "type": "query",
3  "channelName": "ch1",
4  "chaincodeId": "ccsdn",
5  "fcn": "getDeviceInfo",
6▼  "result": {
7    "dvcID": "10.10.10.100",
8▼    "hashes": [
9▼      {
10     "hashID": "1kryIbFfcxraifnJxt84I3kdGKJ",
11     "dvcID": "10.10.10.100",
12     "path": "/etc/passwd",
13     "value": "fn3y09bty340c0rf354tv4635bv37c3f04n5v2v5204ocweoi"
14   }
15 ],
16  "status": true
17 }
18 }

```

| | |
|------------------|----------|
| Test Case Result | Achieved |
|------------------|----------|

Table 37: BIAD_04 unit test

| | | | |
|------------------|--|-----------|----------------------------|
| Test Case ID | BIAD_04 | Component | BIAD network and chaincode |
| Description | The device updates the hash and the parameter registered on the blockchain, so it can be processed. This time we will introduces an anomaly performance. | | |
| Req ID | FR-UR-06;FR-GR-06; FR-UC4-05 | Priority | High |
| Prepared by | TECNALIA | Tested by | TECNALIA |
| Pre-condition(s) | <ul style="list-style-type: none">Each component of the BIAD has to be up and listening on their respective port, which includes at least an orderer (7050), a peer (7051), the CA (7054), couchdb (5084) and REST API (4040).The chaincode must be installed and instantiated on the blockchain network.The device must have a way to sign a transaction. In this case it is done via Rest API, so it must be log in.The associated device must be previously registered on the blockchain and have a devID identifier. And in case of processing parameter, it must be previously registered as well. | | |
| Test steps | | | |
| 1 | The device sends a POST HTTP request to the API, using UpdateHash or UpdateParam methods, with their respective identity and actual values in the body, in JSON format. | | |
| 2 | The orderer receives and mined the transaction in the next block, so that it can be added to the chain. | | |
| 3 | Check if the Hash/Parameter has been correctly updated and if there is any alert, using GetDeviceInfo route at the API. | | |
| Input data | UpdateHash: {"hashID":"1krylbFfcxraifnJxt84I3kdGKJ", "value":"attack_to_10.10.10.100"} UpdateParam: | | |

| | |
|--------|---|
| | <pre>{"recordID":"1kryIbFfcxraifnJxt84I3kdGKJ", "value": 80} {"dvcID":"10.10.10.100"}</pre> |
| Result | <ul style="list-style-type: none"> Step 1 <div> POST cpd_url / channel / chaincode / invoke/updateHash Send </div> <div> JSON Bearer Query Header 1 Docs <pre>1 { 2 "hashID": "1kryIbFfcxraifnJxt84I3kdGKJ", 3 "value": "attack_to_10.10.10.100" 4 }</pre> <div>Beautify JSON</div> <div> 200 OK 2.36 s 399 B Just Now </div> <div> Preview Header 12 Cookie Timeline <pre>1 { 2 "type": "invoke", 3 "channelName": "ch1", 4 "chaincodeId": "ccsdn", 5 "fcn": "updateHash", 6 "transactionID": 7 "5c588d29bbd4e65135337f0dd043e91fc884b0c980fa66947642d060a06f4bf5", 8 "result": { 9 "alertID": "1ks04kNTuVzIFh8lquRqoRzzm1h", 10 "dvcID": "10.10.10.100", 11 "hashID": "1kryIbFfcxraifnJxt84I3kdGKJ", 12 "class": "Corrupted hash", 13 "prevValue": 14 "fn3y09bty340c0rf354tv4635bv37c3f04n5v2v5204ocweoi", 15 "rcvValue": "attack_to_10.10.10.100" 16 } 17 }</pre> </div> </div> Step 2 |

```

2020-11-27 10:43:22.053 UTC [dispatcher] Debug -> DEBU ba1 invoke() called
2020-11-27 10:43:22.054 UTC [dispatcher] Debug -> DEBU ba2 initializing ledgerbuilder internal pools
2020-11-27 10:43:22.054 UTC [dispatcher] Debug -> DEBU ba3 runner stub runtime detected as: "fabric.ChaincodeStub
2020-11-27 10:43:22.054 UTC [dispatcher] Debug -> DEBU ba4 initializing ledgerbuilder internal pools for production environment
2020-11-27 10:43:22.055 UTC [dispatcher] Debug -> DEBU ba5 creating new smart stub for hyperledger fabric
2020-11-27 10:43:22.055 UTC [dispatcher] Debug -> DEBU ba6 fetching requested operation in from current chaincode runner...
2020-11-27 10:43:22.055 UTC [dispatcher] Info -> INFO ba7 getting current operation id from stub encoded data
2020-11-27 10:43:22.056 UTC [dispatcher] Info -> INFO ba8 received argument count is: 2
2020-11-27 10:43:22.056 UTC [dispatcher] Info -> INFO ba9 Requested operation key: updateHash
2020-11-27 10:43:22.056 UTC [dispatcher] Info -> INFO baa decoding chaincode stub payload content
2020-11-27 10:43:22.056 UTC [dispatcher] Info -> INFO bab Received data (via stub) length: 73
2020-11-27 10:43:22.057 UTC [dispatcher] Info -> INFO bac Received data (via stub) data is: {"hashID":"1kryIbFfcxraifnJxt84I3kdGKJ","value":
"attack_to_10.10.10.100"}
2020-11-27 10:43:22.057 UTC [dispatcher] Info -> INFO bad Received chaincode op key: updateHash
2020-11-27 10:43:22.057 UTC [dispatcher] Debug -> DEBU bae using operation stub decoder...
2020-11-27 10:43:22.058 UTC [controller] Debug -> DEBU baf decoding stub using controller stub reader...
2020-11-27 10:43:22.058 UTC [controller] Debug -> DEBU bb0 decoding stub using controller data structure...
2020-11-27 10:43:22.058 UTC [ccsdn] Debug -> DEBU bb1 creating new hash
2020-11-27 10:43:22.059 UTC [dispatcher] Info -> INFO bb2 Parsed data from stub is: {"hashID":"1kryIbFfcxraifnJxt84I3kdGKJ","value":"attack_
to_10.10.10.100"}
2020-11-27 10:43:22.059 UTC [dispatcher] Debugf -> DEBU bb3 detected data is of type: "model.HashModel
2020-11-27 10:43:22.059 UTC [dispatcher] Debug -> DEBU bb4 Applying data preprocessing middlewares
2020-11-27 10:43:22.059 UTC [middleware] Info -> INFO bb5 Checking asset against schema index = 0 middleware = request-not-nil
2020-11-27 10:43:22.059 UTC [ccsdn] Info -> INFO bb6 Checking asset is not nil or empty
2020-11-27 10:43:22.059 UTC [ccsdn] Info -> INFO bb7 checking hashes
2020-11-27 10:43:22.064 UTC [controller] Debug -> DEBU bb8 Reading asset with ID "DeviceModel.10.10.10.100
2020-11-27 10:43:22.067 UTC [controller] Debug -> DEBU bb9 Readed data length: 184
2020-11-27 10:43:22.068 UTC [controller] Debug -> DEBU bba Readed raw data is: {"dvcID":"10.10.10.100","hashes":[{"dvcID":"10.10.10.100","h
ashID":"1kryIbFfcxraifnJxt84I3kdGKJ","path":"/etc/passwd","value":"fn3y09bty340c0rf354tv4635bv37c3f04n5v2v5204ocweoi"}],"status":true}
2020-11-27 10:43:22.068 UTC [ccsdn] Notice -> NOTI bbb Received hash does not match
2020-11-27 10:43:22.068 UTC [controller] Debug -> DEBU bbc Reading asset with ID "DeviceModel.10.10.10.100
2020-11-27 10:43:22.071 UTC [controller] Debug -> DEBU bbd Readed data length: 194
2020-11-27 10:43:22.071 UTC [controller] Debug -> DEBU bbe Readed raw data is: {"dvcID":"10.10.10.100","hashes":[{"dvcID":"10.10.10.100","h
ashID":"1kryIbFfcxraifnJxt84I3kdGKJ","path":"/etc/passwd","value":"fn3y09bty340c0rf354tv4635bv37c3f04n5v2v5204ocweoi"}],"status":true}
2020-11-27 10:43:22.072 UTC [ccsdn] Info -> INFO bbf Updating device
2020-11-27 10:43:22.073 UTC [controller] Debug -> DEBU bc0 key split algorithm: disabled
2020-11-27 10:43:22.073 UTC [controller] Info -> INFO bc1 Saving asset with ID : "DeviceModel.10.10.10.100
2020-11-27 10:43:22.073 UTC [controller] Info -> INFO bc2 Saving asset with DATA : {"dvcID":"10.10.10.100","hashes":[{"hashID":"1kryIbFfcxr
aifnJxt84I3kdGKJ","dvcID":"10.10.10.100","path":"/etc/passwd","value":"attack_to_10.10.10.100"}],"alerts":[{"alertID":"1ks04kNTuVzIFh8lquRqoRzzmlh",
"hashID":"10.10.10.100","hashID":"1kryIbFfcxraifnJxt84I3kdGKJ","class":"Corrupted hash","prevValue":"fn3y09bty340c0rf354tv4635bv37c3f
04n5v2v5204ocweoi","rcvValue":"attack_to_10.10.10.100"}],"status":true}
2020-11-27 10:43:22.074 UTC [controller] Debug -> DEBU bc3 emitting put state event
2020-11-27 10:43:22.074 UTC [ccsdn] Log -> DEBU bc4 Hash updated
2020-11-27 10:43:22.075 UTC [dispatcher] Info -> INFO bc5 Returning result to client via stub 200 {"alertID":"1ks04kNTuVzIFh8lquRqoRzzmlh",
"dvcID":"10.10.10.100","hashID":"1kryIbFfcxraifnJxt84I3kdGKJ","class":"Corrupted hash","prevValue":"fn3y09bty340c0rf354tv4635bv37c3f04n5v2v52
04ocweoi","rcvValue":"attack_to_10.10.10.100"}

```

• Step 3

| | |
|------------------|--|
| Test Case Result | <div> <div>POST cpd_url / channel / chaincode / query/getDeviceInfo Send</div> <div> <div>JSON ▼</div> <div> <div>Bearer ▼</div> <div>Query</div> <div>Header 1</div> <div>Docs</div> </div> </div> <pre> 1 { 2 "dvcID": "10.10.10.100" 3 } </pre> <div>Beautify JSON</div> <div> <div>200 OK</div> <div>241 ms</div> <div>497 B</div> <div>Just Now ▼</div> </div> <div> <div>Preview ▼</div> <div>Header 12</div> <div>Cookie</div> <div>Timeline</div> </div> <pre> 1 { 2 "type": "query", 3 "channelName": "ch1", 4 "chaincodeId": "ccsdn", 5 "fcn": "getDeviceInfo", 6 "result": { 7 "dvcID": "10.10.10.100", 8 "hashes": [9 { 10 "hashID": "1kryIbFfcxraifnJxt84I3kdGKJ", 11 "dvcID": "10.10.10.100", 12 "path": "/etc/passwd", 13 "value": "attack_to_10.10.10.100" 14 } 15], 16 "alerts": [17 { 18 "alertID": "1ks04kNTuVzIFh8lquRqoRzzm1h", 19 "dvcID": "10.10.10.100", 20 "hashID": "1kryIbFfcxraifnJxt84I3kdGKJ", 21 "class": "Corrupted hash", 22 "prevValue": 23 "fn3y09bty340c0rf354tv4635bv37c3f04n5v2v5204ocweoi", 24 "rcvValue": "attack_to_10.10.10.100" 25 } 26], 27 "status": true 28 } </pre> </div> |
| | Achieved |

Table 38: Lib_mon_001 unit test

| | | | |
|------------------|---|-----------|------------|
| Test Case ID | Lib_mon_001 | Component | BIAD Agent |
| Description | Testing of the Monitoring Lib | | |
| Req ID | FR-GR-06; FR-UC4-05 | Priority | -- |
| Prepared by | TECNALIA | Tested by | TECNALIA |
| Pre-condition(s) | <ul style="list-style-type: none">• Source code of the monitor_lib• No built code | | |
| Test steps | | | |
| 1 | cd lib/monitor_lib | | |
| 2 | make test | | |
| 3 | cd test && ./test | | |
| Input data | | | |
| Result | <div>prt0875a@PRT0875A: ~/Proyectos/SDNmicrosense/agent-sdn/lib/monitor_lib/test</div> <div>Archivo Editar Ver Buscar Terminal Ayuda</div> <div>prt0875a@PRT0875A: ~/Proyectos/SDNmicrosense/agent-sdn/lib/monitor_lib/test\$./test</div> <div>Number of active processes: 777</div> <div>USED RAM memory in porcentaje: 98.932030</div> <div>Uptime in seconds: 398</div> | | |
| Test Case Result | Achieved | | |

Table 39: Lib_hash_001 unit test

| | | | |
|------------------|--|-----------|------------|
| Test Case ID | Lib_hash_001 | Component | BIAD Agent |
| Description | Testing of the Hashing Lib | | |
| Req ID | FR-GR-06; FR-UC4-05 | Priority | -- |
| Prepared by | TECNALIA | Tested by | TECNALIA |
| Pre-condition(s) | <ul style="list-style-type: none">• Source code of the hashlib• No built code | | |
| Test steps | | | |

| | |
|------------------|--|
| 1 | cd lib/hashlib |
| 2 | make test |
| 3 | cd test && ./test |
| Input data | |
| Result | <pre> prt0875a@PRT0875A: ~/Proyectos/SDNmicrosense/agent-sdn/lib/hashlib/test Archivo Editar Ver Buscar Terminal Ayuda prt0875a@PRT0875A: ~/Proyectos/SDNmicrosense/agent-sdn/lib/hashlib/test -- master -- ./test 5dd39cab1c53c2c77cd352983f9641e1 prt0875a@PRT0875A: ~/Proyectos/SDNmicrosense/agent-sdn/lib/hashlib/test -- master -- </pre> |
| Test Case Result | Achieved |

Table 40: Fabric_conn_001 unit test

| | | | |
|------------------|--|-----------|------------|
| Test Case ID | Fabric_conn_001 | Component | BIAD Agent |
| Description | Testing of the Fabric connection Lib | | |
| Req ID | FR-GR-06; FR-UC4-05 | Priority | -- |
| Prepared by | TECNALIA | Tested by | TECNALIA |
| Pre-condition(s) | <ul style="list-style-type: none">• Source code of the fabric_conn• No built code | | |
| Test steps | | | |
| 1 | cd lib/fabric_conn | | |
| 2 | make test | | |
| 3 | cd test && ./test | | |
| Input data | | | |

| | |
|------------------|-----------------------------------|
| Result | Terminal does not show any error. |
| Test Case Result | Achieved |

Table 41: agent_001 unit test

| | | | |
|------------------|--|-----------|------------|
| Test Case ID | agent_001 | Component | BIAD Agent |
| Description | Testing of the agent. | | |
| Req ID | FR-GR-06; FR-UC4-05 | Priority | -- |
| Prepared by | TECNALIA | Tested by | TECNALIA |
| Pre-condition(s) | <ul style="list-style-type: none">• Source code of the agent• No built code | | |
| Test steps | | | |
| 1 | cd agent | | |
| 2 | make agent | | |
| 3 | ./agent (with root permissions) | | |
| Input data | | | |
| Result | Terminal does not show any error. | | |
| Test Case Result | Achieved | | |

6. System Installation

6.1 E-auction module

6.1.1 E-auction module installation

To deploy the e-auction module, first of all it is necessary to have an operative Hyperledger Fabric blockchain network and create a channel with the participants. For the proof of concept, we have launched a network using Docker containers, that is based on the basic network that Hyperledger Fabric provides. We have added 3 more organizations to simulate a network with 3 prosumers and 1 ESCO company, because the basic network includes only 1 organization and 1 peer node.

Requirements for the Hyperledger Fabric blockchain network and API:

- Docker
- Go language

Commands for starting the network and the applications for each organization:

1. `cd basic-network` from root project folder
2. `./start.sh` and wait until Fabric network is setup
3. `cd OrgXApp` from root folder
4. `./blocktion`
5. `cd TokenBankApp` from root folder
6. `./blocktion`

6.2 Blockchain Based Intrusion and Anomaly Detection module

6.2.1 BIAD installation

To deploy the BIAD, first of all it is necessary to have an operative Hyperledger Fabric blockchain network and create a channel with the participants. For the proof of concept, we have launched a network using Docker containers.

Requirements for the Hyperledger Fabric blockchain network and API:

- npm
- docker
- nvm v8.17.0

Commands for deployment the Hyperledger Fabric blockchain network and API:

- `yarn generate`
- `yarn rest:start`
- `yarn explorer:start`

Then, we install and instantiate the chaincode, named *ccsdn* for the example, on the created channel.

Requirements for chaincode component:

- fabric-tecnalia-cli
- network-profile

Commands for the deployment of chaincode component:

- `install -f ./network/cc.network-profile.yaml --channel ch1 -c [chaincodeName] -g ./ -p [chaincodePath] -t golang -V`
- `instantiate -f ./network/cc.network-profile.yaml --channel ch1 -c [chaincodeName] -V`

Finally, we need to deploy the agent on the client (see section 6.2.2) and create a certificate on the CA, which will be used to sign the transactions. The client will send input data to the REST API server, where the transaction will be generated and sent to the blockchain corresponding node.

6.2.2 Agent installation

Before any installation of the agent can take place, some prerequisites must be met (in the Project Raspberry PI simulating the devices are used to install the agent). To do so, the script *install_prerequisites.sh* must be run. Once the prerequisites are installed it is time to install the agent.

The code provided contains a Makefile, which compiles all the software by writing *make all*. When doing *make all*, the next parts are compiled:

- Hash_lib: Library which hashes files
- Monitor_lib: Library which gets information about RAM, processes and uptime
- Fabric_lib: Library which connects with the Blockchain module
- Processhider_lib: Library which implements the process hider utility.
- Agent: executable of the BIAD agent

Before doing any *make all* command, it is required to configure the endpoint with the Blockchain, the files to be hashed and the ID of the device sending information. This is directly made in the agent.c file, by changing the variables. Here we have an example of a possible configuration, with the information to be modified in red color:

```
#define num_files 2
char deviceID[20]="1111";
char blockchain_node[20]="172.26.41.127";

char files[num_files][length_files]=
{
    "/etc/passwd",
    "/var/log/auth.log"
};
```

Once this information has been updated, the *make all* command will install the whole software. Also, for testing purposes, a *test/* folder is created within every library folder with code testing the functionality when *make all* is launched. This allows to test every single part of the agent before being integrated into the final platform.

7. Innovation Summary

As part of task T4.5, a complete and secure-by-design Blockchain-based Energy Trading Framework has been designed and developed as a part of the SDN-microSENSE project. The Blockchain-based Energy Trading Framework leverages the other modules of SDN-microSENSE to provide functionalities that enhance the robustness and efficiency of the grid. Specifically, any potential trades that have been calculated by the energy trading framework are further checked by the OTSC tool in terms of feasibility and with regards to the balance of the system, before they are actually applied to the grid. Furthermore, an integrated, distributed IDS (BIAD) have been developed to work complementary to the market that checks the validity of the participant's devices. BIAD communicates with XL-EDPS and S-RAF providing another layer of security to the system and ensures that the participants are trusted and pose no risk to the overall operation of the electrical network. To our knowledge this complete approach regarding energy blockchain market have not been implemented in such a wide manner,

encompassing trading, security, privacy and performance at the same time with regards to the whole system.

One of the innovations is the use of the Hyperledger DLT open source project, in this case, Hyperledger Fabric. Hyperledger Fabric is better suited for the EPES domains in terms of performance [3] and data privacy in contrast to other initiatives which are based on other blockchain platforms. By relying on a permissioned architecture, security and privacy of the e-auction transactions are ensured towards the peers in the network in contrast to public blockchain technologies. On that aspect, the e-auction module is based on the use of smart contracts, providing an automated and decentralized way of resolving auctions and uses a Vickrey-Clark-Groves (VCG) auction model. VCG is a sealed-bid type of auction that ensures maximization of utility, as there could be multiple winners and the system would assign the sold energy to buyers in a socially optimal manner [4]. At the same time, VCG pushes the buyers to bid near their true estimated value. Additionally, incentives are provided to drive participants to participate in a regular manner in order to climb the priority table. Penalties ensure that participants will deliver the agreed amount of energy and will pay their debts, as non-compliance of these two factors would lead to a fall in the priority table. Finally, a bank mechanism protects any participants that is owed tokens.

The second main innovation is the integration of security monitoring and reporting procedures in the energy trading framework by the BIAD module. The BIAD module is a distributed system comprising of lightweight agents, which are installed on the devices of the participants (i.e. RPI based smart meter, IEDs, RTUs). The agent calculates a hash from critical files of the system at regular intervals and uses the blockchain DLT to ensure the uncompromised status of the devices by checking for any difference with the previously reported hash. In case of any anomaly, the owner of the device cannot participate in any auctions; either as seller or as bidder and XL-EDPS is informed for further actions. One of the main advantages of the BIAD agents is that they are lightweight and are written in C, which enables support to any hardware in the market, while at the same time ensuring high-performance. The installation and any configuration of the agent is made locally, as such no information is disseminated outside of the device, other than the hash, further preserving the privacy of the device. Additionally, the agent can be equipped with process-hiding mechanisms in order to inhibit any attacker from discovering and tampering with the agent itself. Overall, BIAD is an innovative IDS, leveraging the distributed blockchain architecture to provide an easy indication of trust among participating devices and operating as another layer of defense against cyber-attacks.

8. Conclusions

In conclusion, this deliverable describes the final outcome of Task T4.5, Energy Exchange using Blockchain Technologies, which is included in WP4. Some minor updates are possible by the end of the project, in the context of the continuous improvement of the complete developed system. These updates will be mainly related to the algorithms run by the software agents participating in the Energy Trading Framework, and the further expansion of the Framework's functionalities.

The Energy Trading Framework has been implemented and its functionalities, inputs/outputs and interconnections with other SDN-microSENSE components and modules have been analytically described, along with a presentation of all the related work in the literature. Taking under consideration the project's requirements (D2.2) and its architecture (D2.3), the system was developed using the Hyperledger Fabric framework. The system offers a safe and privacy preserving environment, in which energy transactions can be performed between prosumers of an islanded part of the main grid. Thus, the island can be sustainable and fully operational until it is ready to reconnect to the main grid. Moreover, the cryptographic mechanisms which were deployed ensure the participants' sensitive data privacy, and the algorithms run by their corresponding software agents, ensure their economic sustainability.

This deliverable produces the fourth layer of the SDN-SELF component, which aims to improve the robustness of the grid against cyberattacks. The blockchain-based Energy Trading Framework offers increased resilience against cyber threats, taking into account the real time security status of the network. The continuous monitoring of the grid's infrastructure and the filtering of all the proposed transactions to verify their feasibility, provides secure energy trading transactions between microgrids that do not jeopardize the grid's stability.

The adoption of an ERC20 token mechanism for financial transactions of trading parties, overcomes the computation overload that is imposed by cryptocurrency such as Bitcoin or Ether and makes transactions faster and more efficient. Also, the utilization of a private permissioned network that Hyperledger Fabric provides, enhances transaction privacy through dedicated channels for energy trading between organizations and fosters an ecosystem of trusted parties that have competitive interests but common goals. Finally, the selection of Vickrey Clarke-Groves auction mechanism, keeps the benefits of simple Vickrey auction mechanism that other studies suggest, such as sealed bids that ensure the true valuation of the product and adds extra value by sharing the offered amount of energy to multiple winners according to their needs, maximizing the value of the product at the same time.

The system presented in this document refers to islands not connected to the main distribution grid. These islands consist of prosumers, which have the ability to produce and store energy, apart from consuming it. The OTSC tool ensures the energy sustainability of the islands upon their initial creation. A possible improvement of the Energy Trading Framework is the addition of the functionality of trading energy with the main grid. In this case, consumers without the ability to produce or store energy can be included to the energy market by participating in energy auctions as bidders. Also, incentives will be given to the consumers, for example, for having lower consumption at peak hours of the day. All the stakeholders will receive the prices offered by the main grid for buying and selling energy, and they will compare these prices with the corresponding prices offered by the rest of the stakeholders. This way, they will have the ability to choose the market which ensures them the maximum profit and at

the same time, the local, blockchain-based energy market will operate in alignment with the main grid's market.

The evaluation of the Energy Trading Framework during the implementation of the SDN-microSENSE pilot demonstrations will lead to further development and research regarding this component, and the possible improvement of its functionalities will be investigated. Another possible extension that may be explored, is the implementation of a web interface that provides suggestions to end users for participation to the market, instead of the current fully automated procedures that are processed by software agents.

References

- [1] "hyperledger.org," Hyperledger, [Online]. Available: https://www.hyperledger.org/wp-content/uploads/2020/03/hyperledger_fabric_whitepaper.pdf. [Accessed 08 12 2020].
- [2] P. G. Sessa, N. Walton and M. Kamgarpour, "Exploring the Vickrey-Clarke-Groves Mechanism for Electricity Markets," *IFAC-PapersOnLine*, vol 50, issue 1, pp. 189-194, 2017.
- [3] M. Dabbagh, M. Kakavand, M. Tahir and A. Amphawan, "Performance Analysis of Blockchain Platforms: Empirical Evaluation of Hyperledger Fabric and Ethereum," in *2020 IEEE 2nd International Conference on Artificial Intelligence in Engineering and Technology (IICAET)*, Kota Kinabalu, Malaysia, 2020.
- [4] P. G. Sessa, N. Walton and M. Kamgarpour, "Exploring the Vickrey-Clarke-Groves Mechanism for Electricity Markets," *IFAC-PapersOnLine*, Vol.50, Issue 1, pp. 189-194, July 2017.
- [5] J. Wang, Q. Wang, N. Zhou and Y. Chi, "A novel electricity transaction mode of microgrids based on blockchain and continuous double auction," *Energies*, vol 10, p. 1971, November 2017.
- [6] M. Andoni, V. Robu, D. Flynn, S. Abram, D. Geach, D. Jenkins, P. McCallum and A. Peacock, "Blockchain technology in the energy sector: A systematic review of challenges and opportunities," *Renewable and Sustainable Energy Reviews*, Elsevier vol 100, pp. 143-174, February 2019.
- [7] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2009.
- [8] P. Vytelingum, "The structure and behaviour of the Continuous Double Auction," 2006.
- [9] A. Hahn, R. Singh, C. C. Liu and S. Chen, "Smart contract-based campus demonstration of decentralized transactive energy auctions," in *IEEE Power and Energy Society Innovative Smart Grid Technologies Conference, ISGT*, 2017.
- [10] V. Buterin, "Ethereum Whitepaper," 9 October 2020. [Online]. Available: ethereum.org.
- [11] L. M. Ausubel and P. Milgrom, "The Lovely but Lonely Vickrey Auction," in *Combinatorial Auctions*, 2005.
- [12] J. Guerrero, A. C. Chapman and G. Verbic, "Decentralized P2P Energy Trading Under Network Constraints in a Low-Voltage Network," *IEEE Transactions on Smart Grid*, vol. 10, pp. 5163-5173, September 2019.
- [13] Z. Li, J. Kang, R. Yu, D. Ye, Q. Deng and Y. Zhang, "Consortium blockchain for secure energy trading in industrial internet of things," *IEEE Transactions on Industrial Informatics*, pp. 3690 - 3700, August 2018.

-
- [14] K. Gai , Y. Wu, L. Zhu, M. Qiu and M. Shen, "Privacy-Preserving energy trading using consortium blockchain in smart grid," *IEEE Transactions on Industrial Informatics*, vol. 15, pp. 3548-3558, June 2019.
- [15] M. A. Ferrag and L. Maglaras, "DeepCoin: A Novel Deep Learning and Blockchain-Based Energy Exchange Framework for Smart Grids," *IEEE Transactions on Engineering Management* vol. 67, pp. 1285-1297, November 2020.
- [16] S. Wang, A. F. Taha, J. Wang, K. Kvaternik and A. Hahn, "Energy Crowdsourcing and Peer-to-Peer Energy Trading in Blockchain-Enabled Smart Grids," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 49, pp. 1612-1623, June 2019.
- [17] A. S. Yahaya, N. Javaid, F. A. Alzahrani, A. Rehman, I. Ullah, A. Shahid and M. Shafiq, "Blockchain Based Sustainable Local Energy Trading Considering Home Energy Management and Demurrage Mechanism," *Sustainability*, vol 12, pp. 1-28, 21 2020 April.
- [18] D. Han, C. Zhang, J. Ping and Z. Yan, "Smart contract architecture for decentralized energy trading and management based on blockchains," *Energy*, vol. 199, 15 May 2020.
- [19] R. Chaudhary, A. Jindal, G. S. Aujla, S. Aggarwal, N. Kumar and K. R. Choo, "BEST: Blockchain-based secure energy trading in SDN-enabled intelligent transportation system," *Computers&Security*, pp. 288-299, Augoust 2019.
- [20] "Blockchain-Based Distributed Energy Trading in Energy Internet: An SDN Approach," *IEEE Access*, vol. 7, pp. 173817-173826, 2 December 2019.